TAMPERE UNIVERSITY OF TECHNOLOGY

ESKO PEKKARINEN

WIRELESS SENSOR NETWORK SIMULATION WITH OMNeT++

Master of Science Thesis

Examiners: Adjunct Prof. Marko Hännikäinen, Dr. Tech. Jukka Suhonen
Examiners and topic approved in the Computing and Electrical Engineering Faculty Meeting on 8th November 2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Electrical Engineering
PEKKARINEN, ESKO: Wireless Sensor Network Simulation with OMNeT++
Master of Science Thesis, 63 pages
April 2014
Examiners: Adjunct Prof. Marko Hännikäinen, Dr. Tech. Jukka Suhonen
Keywords: Network simulation, Wireless sensor networks

Wireless sensor networks (WSNs) are an emerging technology for a broad spectrum of applications. A WSN can consist even hundreds of thousands of devices measuring, controlling and relaying the collected date. The devices often have to endure harsh environments and operate unattended for long periods of time. Wireless communication and energy-awareness are mandatory to meet these objective. It is evident that the design of WSNs is challenging.

Simulation tools enables fast exploration of the desing space and comparison of options in early design phases. In WSN research simulation can be applied to performance estimation of the communication protocols, device functionality and applications. Most of the current network simulators focus on Internet protocols and are not viable for WSN simulation. All the essential aspects of WSNs must be accounted for in the simulation.

This Thesis discusses the applicability of Objective Modular Network Testbed in C++ (OMNeT++) simulation framework for low-power WSN research. Two simulation cases are implemented. The first models a hierarchical wakeup function in a surveillance application. Low-power devices monitor the target area and wake up other devices with video cameras on demand. The simulation shows that hierarchical wake up mechanisms may significantly improve the WSN energy-efficiency. The second simulation evaluates the applicability of a positioning algorihtm in WSNs. It indicates that low-energy positioning in WSN could be feasible using simple distance estimates.

The requirements for a WSN simulator are strict. First, the results must be reliable and accurate. Second, the simulator has to scale with the number of devices in the network with reasonable execution time. Finally, it must be able to model events in both device and network scale. Based on the experience of two simulation cases OMNeT++ proved to be extensible and scalable tool fit for WSN research. However, much like other simulation tools, OMNeT++ is prone to produce too optimistic results.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Sähkötekniikan diplomi-insinöörin tutkinto-ohjelma
PEKKARINEN, ESKO: Wireless Sensor Network Simulation with OMNeT++
Diplomityö, 63 sivua
Huhtikuu 2014
Tarkastajat: professori Marko Hännikäinen, tekniikan tohtori Jukka Suhonen
Avainsanat: Verkkosimulaatio, langattomat sensoriverkot

Langattomat sensoriverkot ovat yleistyvä teknologia monilla sovellusalueilla. Sensoriverkko voi sisältää jopa satojatuhansia laitteita, jotka mittaavat, ohjaavat ja siirtävät keräämäänsä tietoa. Laitteet joutuvat usein toimimaan vaativissa ympäristöissä ja toimimaan pitkiä aikoja ilman huoltotoimia. Langaton tiedonsiirto ja energiatehokkuus ovat välttämättömiä edellytyksiä langattoman sensoriverkon toiminnalle. Nämä vaatimukset tekevät verkkojen suunnittelusta haastavaa.

Simulaatio mahdollistaa suunnitteluavaruuden nopean kartoituksen ja vaihtoehtojen vertailun aikaisessa vaiheessa tuotekehitystä. Langattomien sensoriverkkojen tutkimuksessa simulaatiota voidaan hyödyntää tietoliikenneprotokollien, laitteiden toiminnan ja sovellusten suorituskyvyn arvioinnissa. Suurin osa nykyisistä verkkosimulaattoreista on suunnattu Internetissä käytettäville protokollille ja ovat siten huonosti soveltuvia langattomien sensoriverkkojen simulointiin, joiden erityispiirteet tulee ottaa huomioon käyttökelpoisessa simulaatiossa.

Tässä diplomityössä tarkastellaan Objective Modular Network Testbed in C++ (OMNeT++) simulaattorin soveltuvuutta langattomien sensoriverkkojen simulointiin. Työssä toteutettiin kaksi simulaatiota. Ensimmäisessä mallinnettiin hierarkista herätystoimintoa valvontasovelluksessa. Vähän energiaa kuluttavat laitteet tarkkailevat kohdealuetta sensorein ja havaitessaan liikettä herättävät varsinaiset valvontalaitteet, joihin on asennettu videokamera. Simulaation perusteella herätystoiminto voi merkittävästi parantaa langattoman verkon energiatehokkuutta. Toisessa simulaatiossa tarkastellaan erään paikannusalgoritmin soveltuvuutta langattomissa sensoriverkkoissa. Se osoittaa, että laitteiden paikannuksessa voidaan käyttää yksinkertaisia etäisyysarvioita ilman, että energiankulutus nousee liian korkeaksi.

Vaatimukset langattomien sensoriverkkojen simulaattorille ovat tiukat. Ensinnäkin, simulaation tulosten on oltava luotettavia ja tarkkoja. Toisaalta, sen on skaalauduttava suurelle määrälle laitteita siten, että suoritusaika pysyy kohtuullisena. Lisäksi sillä on kyettävä mallintamaan niin yksittäisen laitteen kuin koko verkon kattavia tapahtumia. Edellä mainittujen kahden simulaation pohjalta OMNeT++ on osoittautunut hyvin laajennettavaksi ja skaalatuvaksi työkaluksi langattomien sensoriverkkojen tutkimuksessa. Monien muiden simulaatiotyökalujen tavoin OMNeT++ kuitenkin tuottaa herkästi liian optimistisia tuloksia.

PREFACE

This Thesis has been written as a part of the wireless sensor network research at Tampere University of Technology.

I want to thank my examiners Marko Hännikäinen and Jukka Suhonen for their guidance and comments during the work. I also want to thank my co-worker Ville Kaseva for the Matlab implementation of the MDS algorithm.

My deepest gratitude to my whole family for unyielding support before, during and after this work.

Tampere, 15th April 2014.

Esko Pekkarinen

TABLE OF CONTENTS

1. Introduction 1
2. Wireless Sensor Networks 4
2.1 Network Topology and Functionality 4
2.2 Protocol Stack 7
2.3 Node Devices 9
2.4 WSN Standards 10
2.4.1 IEEE 802.11 11
2.4.2 IEEE 802.15.4 11
2.4.3 ZigBee 14
2.4.4 Ad hoc On-demand Distance Vector 14
2.4.5 ISO 18000-7 16
2.4.6 DASH7 17
2.5 TUTWSN 17
3. Wireless Sensor Network Simulation 20
3.1 Modeling and Simulation 20
3.2 Discrete Event Simulation 21
3.3 Requirements for WSN Simulator 22
3.4 Network Simulators 23
3.4.1 J-Sim 23
3.4.2 GloMoSim 24
3.4.3 NS-2 24
4. OMNeT++ Simulation Framework 26
4.1 Overview 26
4.2 Modeling Networks 28
4.2.1 Messages 28
4.2.2 Modules and Channels 29
4.3 Simulation Output 31
4.4 Related Work 32
5. WSN Surveillance Simulation 35
5.1 Requirements 36
5.2 Implementation 37
5.2.1 Devices 37
5.2.2 Surveillance Application 38
5.2.3 Routing 38
5.2.4 MAC protocol 43
5.3 Simulation 47
6. Mobile Positioning Network 50

6.1 Positioning Algorithm 50
6.2 Implementation 50
6.3 Simulation 52
6.3.1 Group Walk 52
6.3.2 Stationary Anchor Nodes 54
6.3.3 Constant Speed Walk 56
7. Conclusions 59

LIST OF ABBREVIATIONS

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
ACK	Acknowledge
AODV	Ad hoc On-demand Distance Vector
AP	Access Point
API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application Specific Integrated Circuit
BAN	Body Area Network
BGP	Border Gateway Protocol
CAP	Contention Access Period
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
DES	Discrete Event Simulation
DCF	Distributed Coordination Function
DHCP	Dynamic Host Configuration Protocol
DSDV	Destination-Sequenced Distance-Vector
DSP	Digital Signal Processor
DSR	Dynamic Source Routing
FTP	File Transfer Protocol
DYMO	Dynamic MANET On-demand
FDM	Frequency Division Multiplexing
FEL	Future Event List
FES	Future Event Set
FFD	Full-Functional Device
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPS	Global Positioning System
GTS	Guaranteed Time Slot
HIP	Host Identity Protocol
HTTP	Hypertext Transfer Protocol
HW	Hardware
ICMPv4	Internet Control Message Protocol version 4
ICMPv6	Internet Control Message Protocol version 6
IGMPv2	Internet Group Management Protocol version 2
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LLC	Logical Link Control
LSR	Link State Routing

MAC	Medium Access Control
MCU	Microcontroller Unit
MIPv6	Mobile IPv6
NACK	Not Acknowledged
OMNeT++	Objective Modular Network Testbed in C++
OSPF	Open Shortest Path First
OLSR	Optimized Link State Routing
PAN	Personal Area Network
PCF	Point Coordination Function
PPP	Point-to-Point Protocol
RFD	Reduced Function Device
RFID	Radio Frequency IDentification
RIP	Routing Information Protocol
RSSI	Received Signal Strength Indication
RSTP	Rapid Spanning Tree Protocol
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
SCTP	Stream Control Transmission Protocol
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TTL	Time To Live
TWI	Two-Wire Interface
TUT	Tampere University of Technology
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
WSN	Wireless Sensor Network

1. INTRODUCTION

A Wireless Sensor Network (WSN) is a collection of autonomous devices monitoring a wide physical environment. Each device has sensors for capturing information about its immediate surroundings, but very limited energy reservoir and computational capacity. The network of simple devices will be capable of complex applications, if the devices collaborate. The networks are often deployed in harsh environments where the devices are susceptible to unexpected failure. Wireless communication enables the devices to adjust to changes in the network, if some of the devices fail and become unavailable either temporarily or permanently. Typical application fields are facility management, machine surveillance, health care and military applications [1].

An example of a WSN is shown in Figure 1.1. The WSN is deployed to accomplish a specific task and the devices are positioned to cover the area of the interesting phenomenon. The number of devices in a network can vary from a few up to thousands or even hundreds of thousands [1]. Each device collects data of the monitored phenomenon and reports the findings to a sink [2]. A sink is a device interested in the data and it can either process the data itself or forward it to an external resource. The end-user is mainly interested in the information extracted from the collected data, not the individual devices. In multihop routing networks other devices help it by relaying its data towards the sink.

Simulation is a widely adopted method in analysis, experimentation and training [3]. Every simulation utilizes models. A model captures the behaviour of a single real-world phenomenon. Simulation is essentially observing the model behaviour under known conditions. Estimate of the real world outcome is derived from repeated experiments on the model. The more detailed the models are, the more accurately the estimate is. The amount of detail is often a trade-off with the required amount of effort to create the model. Once the models have been designed, simulation enables fast exploration and comparison of different configurations.

The research and development of WSN algorithms also benefits from simulation. It would be impractical to experiment with a real network prototypes especially if the devices had to be designed and implemented first, or a large number of devices is needed [4]. Simulation is an enabling technology for fast exploration of functionality and parameter combinations for applications and protocols. The objective is to

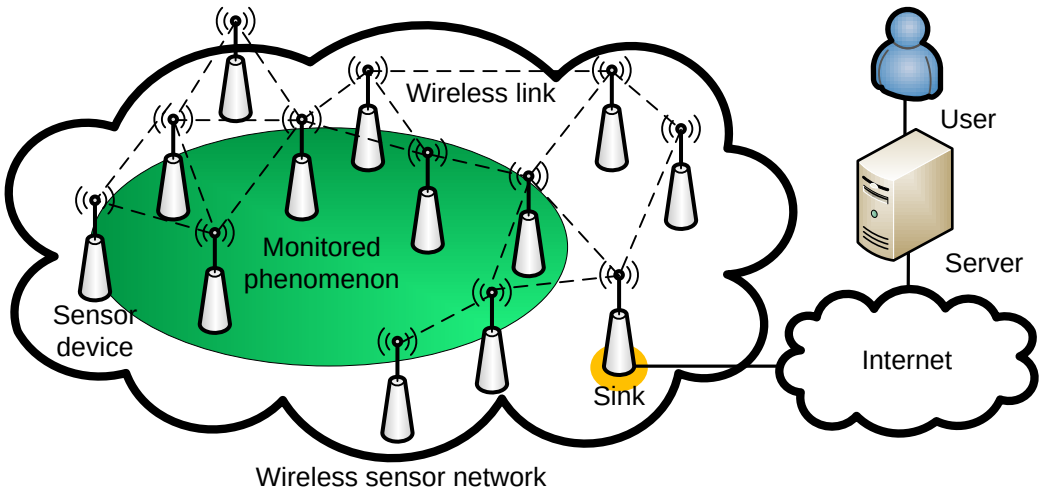


Figure 1.1: The devices in a WSN independently gather data of an interesting phenomenon. The data samples are relayed to the sink device which can forward the data to an external server. The user can access the data on the server without concern of the underlying WSN.

provide early feasibility estimate of an algorithm with less effort, time and cost than what it would take with real devices [5].

In WSNs some of the defining characteristics are application specific configurations, wide scale of network sizes and densities, mobile devices, unreliability of both devices and communication, resource scarcity, tightly coupled interaction with environment, and dynamic changes in the network. A feasible simulation environment must provide means to model these features and scale well with the network size. The majority of currently available network simulators are likely to disregard or oversimplify some of these important aspects in WSNs [5]. Traditionally network simulators have focused on wired networks and support is provided only for wireless protocols designed to be coupled with higher level Internet protocols such as TCP/IP. These protocols are mostly unfeasible in resource-constrained WSNs making a lot of available simulation tools ill-suited for WSN research.

One of the simulators currently used for WSNs is Objective Modular Network Testbed in C++ (OMNeT++) [6]. The main concept of OMNeT++ is that models are encapsulated in modules that interact by sending and receiving messages. The modules can be assembled into a more complex hierarchical structures using compound modules that encapsulate other modules. Each module is customizable, re-usable and easily replaceable with another. The simulation scenario can be altered by changing the module composition.

The Integrated Design Environment (IDE) allows the user to assemble the modules in a graphical editor. The IDE provides all the necessary tools for development, build, simulation configuration, run, visualization and result analysis. The model behaviour and algorithms are described in C++ [7]. For users familiar with C++

learning to use OMNeT++ is fairly straightforward.

While there are many advantages in using OMNeT++ for WSN simulation, the number of available models for relevant protocols is surprisingly low [5]. This increases the background work required for implementing own models and makes the comparative evaluation of new models rather difficult.

The objective of this Thesis is to deploy OMNeT++ in low-power WSN research project. The first step is to setup the simulation environment. Next, two simulation cases are defined and executed. The first simulation case models an existing prototype of a surveillance application implemented earlier in the project. The simulated results are compared with the prototype to evaluate the error introduced by the simulation. The second case models a light-weight positioning algorithm in a WSN. The simulation results are used to evaluate the feasibility of the algorithm on a real WSN platform. Based on the experiences of the two cases, OMNeT++ applicability for WSN research is discussed.

The results of the Thesis are an evaluation and user experience of OMNeT++ in WSN research, analysis of the simulation case results, and the new models utilized in the simulations. The models are a low-power Medium Access Control (MAC) and a routing protocol and a dual protocol stack for devices with two radios. The models are reusable and configurable for future needs.

The structure of the work is as follows: Chapter 2 introduces the field of wireless sensor networks. Chapter 3 discusses the general principles of modeling and simulation and available simulation tools for WSNs. OMNeT++ simulation framework is discussed separately in Chapter 4. Chapters 5 and 6 present the implemented simulation cases, the models they utilize and the simulation results. Finally, Chapter 7 concludes the work.

2. WIRELESS SENSOR NETWORKS

This chapter discusses the general concepts of WSNs. It is important to understand the characteristics of WSNs since the same concepts and issues must be considered in simulation. A plausible simulation environment must provide the means to model these unique features of WSNs.

First, the network functionality is introduced. Next, the required protocols and hardware devices are briefly discussed. A lot of WSN-related standards have been released and those related to the selected simulation cases are discussed in this chapter. Finally, an example of a functional energy-efficient WSN is shown.

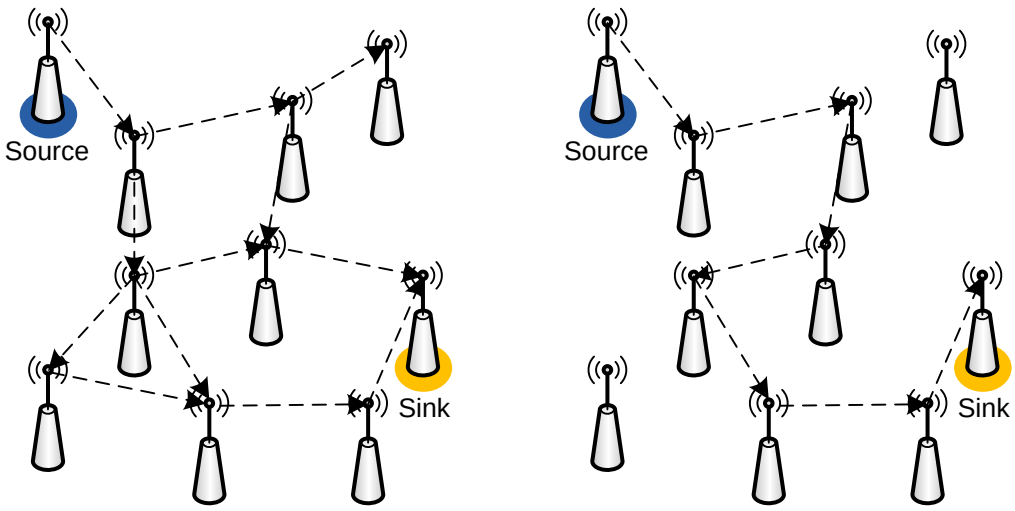
2.1 Network Topology and Functionality

The basic building block of any WSN is a node device. A node is an embedded system designed for sensing surrounding phenomenon, processing the gathered data and communicating the extracted information to the user [1]. When an interesting event occurs in the network area, it is detected by one of the sensor nodes. The node is now the source of the information. Another node is interested in the captured data. This node is called the sink.

To deliver the data to the sink, the source needs a connection to it. In a single-hop communication the sink is reached by the source alone. However, in most networks the majority of nodes are located further away from the sink and cannot reach it directly. Multi-hop communication utilizes other nodes as intermediators who forward the messages from the source towards the sink. A network may have multiple sinks that are interested in different data.

The source and the intermediator nodes must somehow decide, which node they forward the data to so it will eventually arrive at the destination [4]. These forwarding decisions dictate the message route through the network. A routing protocol is responsible for establishing and maintaining these routes from the sources to the sink(s).

The simplest two forwarding rules are flooding and gossiping [4]. In flooding the source floods the network by broadcasting a message to all neighbour nodes who will in turn rebroadcast it. Figure 2.1(a) illustrates the propagation of a data packet from the source node in the upper left corner to the sink node on the right. The packet propagates one hop at a time until all nodes, including the sink, have received



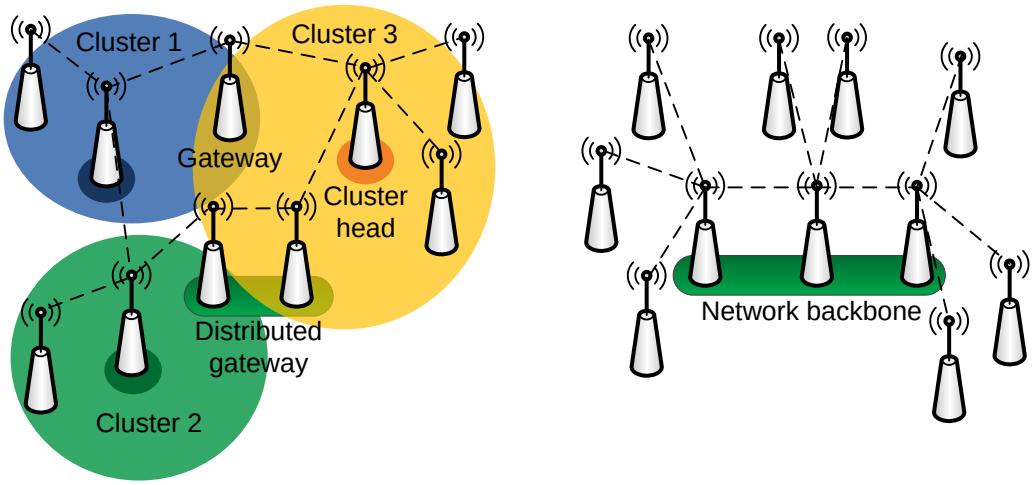
(a) (b)

Figure 2.1: Routing protocol dictates how packets propagate from the source to the sink. Flooding (a) and gossiping (b) are examples of very simple routing protocols.

a copy of the it. Gossiping represents the other extreme: Each node forwards the packet to a random neighbour until it arrives at the destination or until all nodes have seen it. An example of gossiping is shown in Figure 2.1(b). Compared to flooding, gossiping causes less traffic in the network, but the packet delay is likely longer.

The active links between nodes compose the logical topology of the network. In wireless communication the links are prone to errors and interference. A single node can establish a link to any neighbouring node within its transmission range. Yet, it is often beneficial to concentrate the communication to certain neighbours while ignoring the others [1]. This selective communication is called topology control. It is a major concern in most WSNs because of their ad-hoc nature. In ad-hoc networks new nodes may join the network or nodes may leave the network at an arbitrary moment in time. The network must have mechanisms to adapt to these changes.

Network topologies can be divided into two categories: hierarchical and non-hierarchical [1, 8]. In a non-hierarchical (flat) network all nodes have an equal role of keeping track of the changes in the network. A node relies on other nodes to inform it of changes outside its neighbourhood. As the size of the network grows, the amount of messages reporting the changes soon exceed the data messages and the delay increases for both the data and the network state messages. Eventually the network congests when the nodes send more messages than the network can deliver. The surplus messages begin to build a queue at the nodes until the memory constraint of the nodes forces some of the messages to be dropped. As a consequence



(a) (b)

Figure 2.2: Clustering (a) and network backbone (b) are hierarchical network topologies.

the change reported in the dropped message is never delivered to the destination node whose record of the network state now becomes flawed.

In contrast to flat networks, a hierachical network assigns nodes different roles in recording the network status. The two techniques presented here are clustering and network backbone [1]. The main principle in clustering is to divide the network into groups of nodes and assign one node in each group a cluster head. The cluster head controls and manages the resources within the group. In a two-hop cluster each node can communicate with any other node in the cluster either directly or by using the cluster head as a relay. This is often sufficient, but forces small cluster size. However, multi-hop clusters introduce more complexity and may require additional node roles to be used.

An example of a clustered network is shown in Figure 2.2(a). The communication between clusters is handled either by the cluster heads or by gateway nodes. If two cluster heads are more than one node apart, the set of connected nodes between the clusters can form a distributed gateway. It is usually possible for a node to be associated with more than one cluster head at a time.

A hierarchical network may utilize a network backbone instead of clustering. The general concept is similar. A set of connected nodes are selected to form the network backbone. All other nodes are connected to the backbone nodes and all communication is routed through the backbone. An example of a network backbone is shown in Figure 2.2(b).

In any WSN a node may also function as a gateway to another kind of a network, for example the Internet [1]. The user may then receive information from the network or manipulate its operation without a wired connection to each device separately. Moreover, such gateways enable thorough analysis, visualization and utilization of

the measurement data on the external server where the processing capacity is no longer limited by the node devices.

2.2 Protocol Stack

The communication between the nodes follows a chosen protocol that is a set of rules and restrictions for communication. The implementation of such a protocol is fairly complex and therefore modularized for easier design. The modules are then stacked so that the lower layer provides a set of functions for the higher layer who in turn provides more complex functions. The protocol stack in WSNs has typically the following five layers from the highest to the lowest: Application layer, Transport layer, Network layer, Data link layer and Physical layer [2]. The protocol stack and the essential functions of each layer are shown in Figure 2.3.

The application layer describes the interaction of the application software and the protocol stack. All other layers exist only to provide service to it. Since the requirements for the layer are highly application specific, no single protocol or protocols has become widely adopted and application layer remains mostly an unexplored region [2]. Some protocols for example Sensor Management Protocol (SMP), Task Assignment and Data Advertisement Protocol (TADAP) and Sensor Query and Data Dissemination Protocol (SQDDP) have been proposed for use in application layer [2].

The transport layer provides a reliable transport of higher layer packets from one place to another and abstracts away the network [1]. The most important mechanisms for ensuring reliability are packet loss detection and repair. The transport layer should also have mechanisms for avoiding or recovering from network congestion. The transport layer also manages the data flow when the receiver node is temporarily unavailable. These tasks in mind the importance of transport layer is clear when the network can be accessed from an external network, for example the Internet [2]. Node resource scarcity and varying routing protocols in network layer

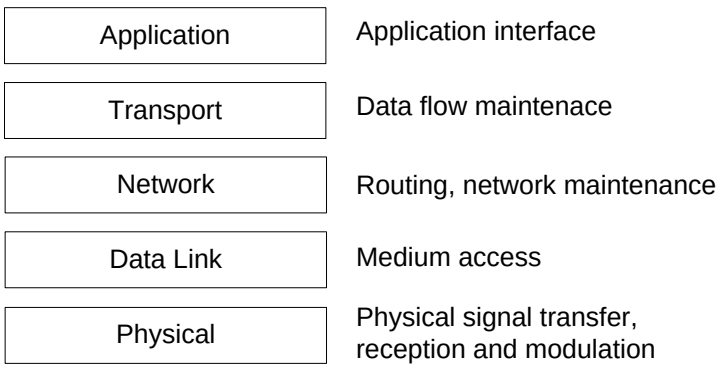


Figure 2.3: The WSN protocol stack has typically five layers.

make traditional transport protocols, for example TCP, mostly unfeasible in WSNs. However, no widespread standards or proposals have yet emerged.

The network layer handles packet routing that is finding a path from the source to the sink. In practice this means that the network structure, links between the nodes, must be created and maintained. Another important function is to provide interworking with external networks such as the Internet [2]. For example, IPv6 over Low power Wireless Personal Area Network (6LoWPAN) [9] enables the transmission of IPv6 [10] packets over IEEE 802.15.4 [11] data link and physical layer. Challenges rise from resource scarity and the unreliability of the wireless medium. Also, energy-efficient consideration in WSNs makes the network layer further complicated.

The data link layer is often divided into two functions: Medium Access Control (MAC) and Logical Link Control (LLC). MAC regulates the access of the nodes to the common transfer medium. Contention-based MAC protocols do not restrict the nodes from accessing the shared medium simultaneously. They target to minimize the number of collisions and provide mechanisms for recovering when collisions do occur. Contention-free protocols secure beforehand that collisions do not happen. Typical contention-free solutions are Frequency Division Multiplexing (FDM) and Time Division Multiple Access (TDMA). They divide either the available frequency band or time into slots and distribute the slots among the nodes. Only one node is allowed to transmit in a given slot. Compared to contention-based MACs the cost is often higher complexity and nodes may have to wait unnecessarily if the reserved slots are not used. The challenge all MAC protocols face is minimizing the power consumption. The single most important method is to avoid idle listening of the medium [1].

LLC hides the underlying physical errors of the wireless medium. The objective is to guarantee a certain level of reliability of transmissions with minimal energy [1]. LLC uses redundancy and retransmissions to improve transfer reliability at the cost of increased energy consumption. Another important function is the management of the links. It should provide mechanisms for link establishment, teardown and estimation of link quality.

Physical layer concerns of the transfer of the data signal, as well as its modulation and demodulation. In WSNs the physical layer should provide a low-power, simple yet robust transfer. To decrease the power consumption small transmission power and low duty cycle must be used. As a consequence, the nodes will have low data rate and small communication range.

2.3 Node Devices

The design and implementation of a single node is an essential part of the whole WSN design since the chosen hardware sets constraints for the network performance. The exact requirements for a single node are highly related to the application at hand, but some general defining characteristics can be derived. The prerequisite for all WSN designs is that the applied technologies must be scalable, adaptive and fault-tolerant [1]. For practical reasons the nodes should also be easily programmable, small of size, have a low production cost and operate unattended for long periods of time [2].

Figure 2.4 illustrates the basic hardware components of a node. Sensors and actuators are used as an interface to the physical world by sensing and controlling nearby events. A controller unit processes these observations and executes the desired actions. Memory provides a storage for the program code defining these actions and all relevant data. A transceiver connects the node to the network by providing means to send and receive messages from other nodes. The power supply provides energy for all components.

A WSN without sensors will not be able to produce any useful information. Sensors provide an interface between the analogous physical world and the digital world of the controller unit [4]. They convert the physical events or quantities into electrical signals that can be digitally processed and stored. The measured signal often requires amplification, filtering or other pre-processing before analog-to-digital conversion (ADC) can be done. Typical sensors in WSNs include thermometers, light sensors, humidity meters, vibration sensors, acceleration sensors, and chemical sensors such as smoke detectors. Actuators provide the controller means to influence the environment. Common methods are opening/closing a switch or relay or controlling a device such as a motor or a light. An actuator should always be coupled with a sensor for feedback [1].

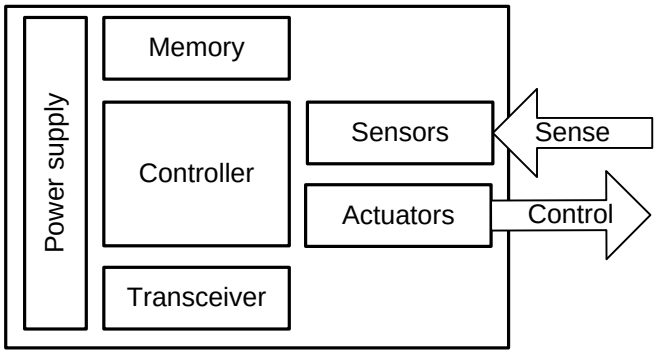


Figure 2.4: A node device has sensors, actuators, a controller, a memory, a transceiver and a power supply. The figure is based on [1].

The controller unit collects and processes the data from sensors, controlling the behavior of actuators and deciding where and when to send the measurement data. The software implementation of the protocol stack contains these decisions. Microcontrollers are particularly designed for embedded systems and often provide many of required features. The desired features may include for example interface modules such as Serial Peripheral Interface (SPI) and Two-Wire Interface (TWI) for sensors, Analog-to-Digital Converter (ADC) modules, internal memories, timers and sleep modes.

Flexibility, low cost, small size and relatively low power consumption make general-purpose microcontrollers a suitable choice for most WSNs. However, for computation-intensive and energy-efficient applications other technologies must be considered. Alternatives include Application Specific Integrated Circuits (ASICs) [12], Digital Signal Processors (DSPs) [13] and Field Programmable Gate Arrays (FPGAs) [14].

Wireless communication is a mandatory prerequisite for WSN operation. Possible technologies are for example optical communication, ultrasound and radio frequencies. Radio frequencies are the most common choice in WSNs. A transceiver device has the capability for transmitting and receiving radio frames. Typical communication over a radio channel is half-duplex and switching between reception and transmission is done according to the MAC protocol. The transceiver is the energy intensive part of the node and therefore it must provide low-energy idle and/or sleep states.

2.4 WSN Standards

Commonly accepted standards ensure that WSN devices from different vendors are interoperable [8]. Devices using the same standardized protocols can be effectively integrated into a single WSN by choosing the best fit device for each task [15]. Furthermore, existing networks can be extended with new devices unconstrained by the current devices in the network. Since the potential of WSNs lies in the large number of devices, this is a significant benefit. Harmonizating pre-existing standards id considered a key challenge in current WSN development [8].

This section presents an overview of WSN communication standards related to this work. IEEE 802.11 [16], Ad hoc On-demand Distance Vector (AODV) [17] and ISO-18000-7 [18] standard have been used in the simulation cases. The simulation models presented in this work are based on the last two. IEEE 802.15.4 [11] and ZigBee [19] are widely adopted in WSNs. DASH7 [20] presents a WSN protocol based on the ISO-18000-7 standard which resembles the MAC model implemented in this work.

2.4.1 IEEE 802.11

The IEEE 802.11 standard is one of the most popular standards for wireless devices [4]. It defines the physical layer and data link layer for Wireless Local Area Networks (WLANs). The frequency band in WLANs is typically 2.4 GHz and the communication range less than 100 m. The maximum data rate for is 100 Mbits/s.

The standard defines two operating modes. In Point Coordination Function (DCF) mode all the communication is routed through a central device known as the Access Point (AP). This forces a star-topology with the AP as the center node which ensures collision-free communication within the network. In Distributed Coordination Function (DCF) each device connects directly with the others. The devices utilize Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) method to prevent collisions. A device will only transmit after listening to the channel and ensuring it is idle. If the channel is not idle, the device will defer its transmission and try again later. Both modes support only single-hop communication.

Although IEEE 802.11 is widely adopted in many application areas, it cannot be applied in WSNs [1, 8]. Both PCF and DCF require a lot of listening to the channel which increases the power consumption to an unacceptable level in WSN nodes. Furthermore, limiting only to a single hop network is impractical for WSNs monitoring wide areas.

2.4.2 IEEE 802.15.4

The IEEE 802.15.4 standard [11] defines a MAC and physical layer for low data rate wireless devices in Personal Area Networks (PANs). The operation frequency band is 868 MHz, 915 MHz or 2.4 GHz and maximum data rates are 20, 40 or 250 kbit/s respectively. A typical operation range of a PAN device is 10 meters.

A devices in the network is categorized as a Full-Function Device (FFD) or a Reduced-Function Device (RFD) [1]. A FFD can assume any of the three device roles: PAN coordinator, coordinator and a device. A PAN coordinator initiates the network and is responsible for its management. Each network has exactly one PAN coordinator. A (PAN) coordinator can communicate with and relay data from any other device in its range. In optional beaconed mode, a coordinator oversees the transmissions of neighbouring devices. A device joins to the network through one (PAN) coordinator in its range and will communicate only with it. A RFD is always an end device and does not have routing capacity. IEEE 802.15.4 uses either 16-bit addresses, which are assigned by the PAN coordinator and unique within the PAN, or extended 64-bit addresses.

In beaconed mode the medium access is controlled by superframes managed by the coordinators [1]. The superframe is divided into an active period and an inactive

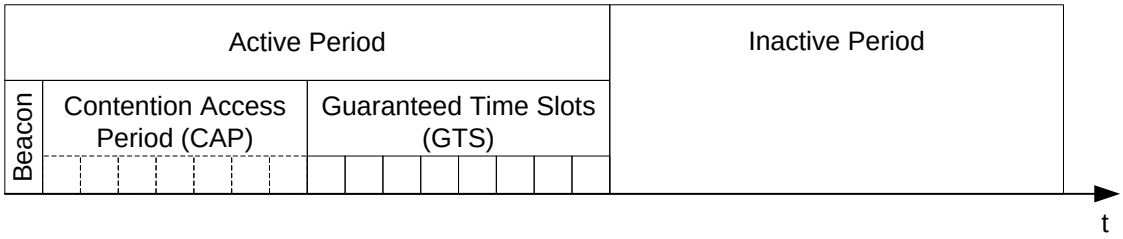


Figure 2.5: IEEE 802.15.4 superframe consists of contention access period, contention-free GTS and an inactive period.

period as shown in Figure 2.5. The active period is further divided into 16 time slots. The first slot is reserved for the beacon packet that describes following the superframe. The remaining 15 slots are distributed into Contention Access Period (CAP) and Guaranteed Time Slots (GTS). During the CAP the devices content for the channel using slotted CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) protocol.

The GTSs are used to assign a device contention-free access to the channel for the duration of the slot. The coordinator announces the assigned slots in the beacon. During the inactive period all devices, including the coordinator, may enter a sleep state to reduce energy consumption. The devices are required to wake up before the beginning of the next superframe to receive the next beacon. During the CAP and GTSs end devices need to wake up only on demand. The coordinator however must listen to the channel for the duration of the CAP and each assigned GTSs. Therefore, the energy consumption in the network is unavoidably asymmetric. The same issue concerns the unbeaconed mode.

In the unbeaconed mode there are no superframes to organize the communication. Devices wake up according to their own schedule to send to or request data from the coordinator using unslotted CSMA-CA protocol [1]. Without synchronization the channel can be accessed at any time so the coordinator has to constantly listen to the channel.

The standard supports two general topologies: star networks and peer-to-peer networks [8, 11]. In a star network, the PAN coordinator is the central node and all communication from other devices, FFD or RFD, must pass through it. An example of a star network is shown in Figure 2.6(a). In a peer-to-peer network, FFDs may communicate directly with others. They can connect in an arbitrary pattern which allows more complex topologies than star-topology. An example of a peer-to-peer network is shown in Figure 2.6(b). A cluster-tree network is a specialized case of peer-to-peer networks. In a cluster-tree the PAN coordinator is the root node, while other FFDs compose the branches and RFDs can join the network as leaves. As the size of the network increases, it is split into clusters. Some of the coordinator FFDs

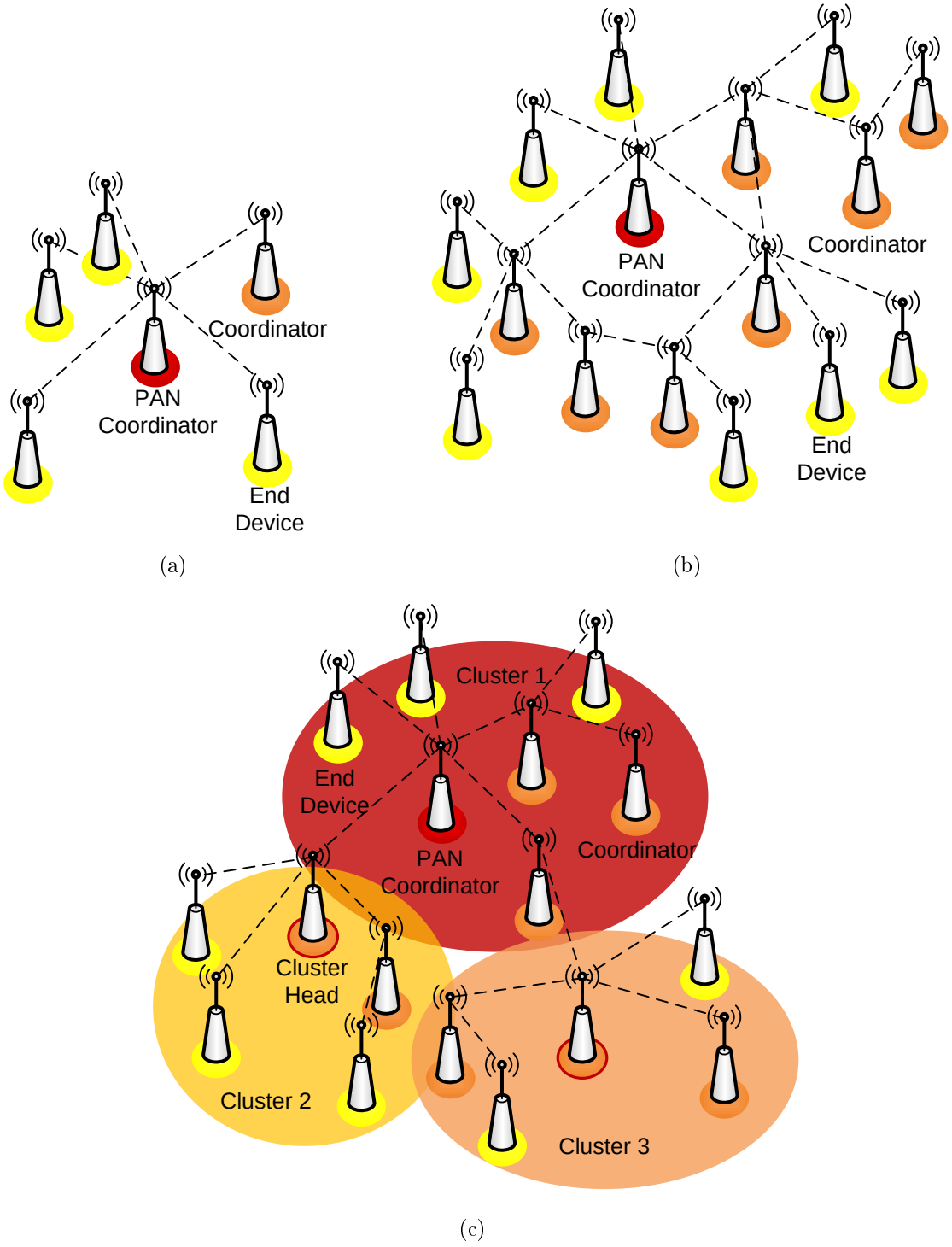


Figure 2.6: The IEEE 802.15.4 standard supports star-topologies (a) and peer-to-peer (b) topologies. Cluster-tree topology (c) is a specialised case of peer-to-peer topology.

become cluster heads to provide synchronization within the cluster. An example is shown in Figure 2.6(c).

Flexibility, low complexity and low power consumption has made IEEE 802.15.4 feasible over a broad spectrum of applications. Peer-to-peer networks allow multi-

hop routing, but since routing is a function of the network layer, it is not included in the original standard. Other standards, such as ZigBee and 6LowPAN, have emerged to complement IEEE 802.15.4 since its ratification in 2003.

2.4.3 ZigBee

The ZigBee Alliance has released IEEE 802.15.4 technology enhanced with higher level protocols. The application layer manages the device role, services and security. The APplication Support sublayer (APS) provides an interface to the network layer. The network layer provides functions for devices to join and leave the network. Multi-hop routing is supported by using AODV routing algorithm.

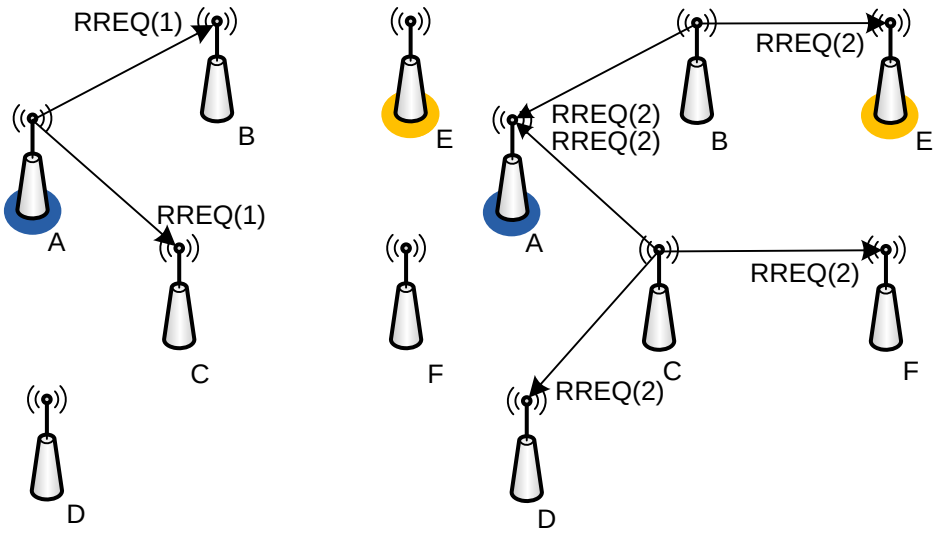
ZigBee has become a widespread technology in embedded systems with high reliability and versatility requirements where low data rates are acceptable. Low data rate and power consumption are inherited from IEEE 802.15.4. Other important features of ZigBee are over two years of lifetime and low complexity of the devices. In heterogeneous networks ZigBee is often the common standard that guarantees interoperability of devices from different vendors. The first version of ZigBee was released in 2004 and updated in 2006.

2.4.4 Ad hoc On-demand Distance Vector

AODV is a light-weight routing protocol commonly found in WSNs and other wireless applications. Each node maintains a private routing table of active routes to other nodes. AODV does not record the complete route, but only the address of the next node on the path towards the destination, the destination address and the minimum number of hops to the destination. When a node wishes to communicate with another, only an active route can be used. If the sender has no route to destination node, it will first initiate a route discovery to establish one.

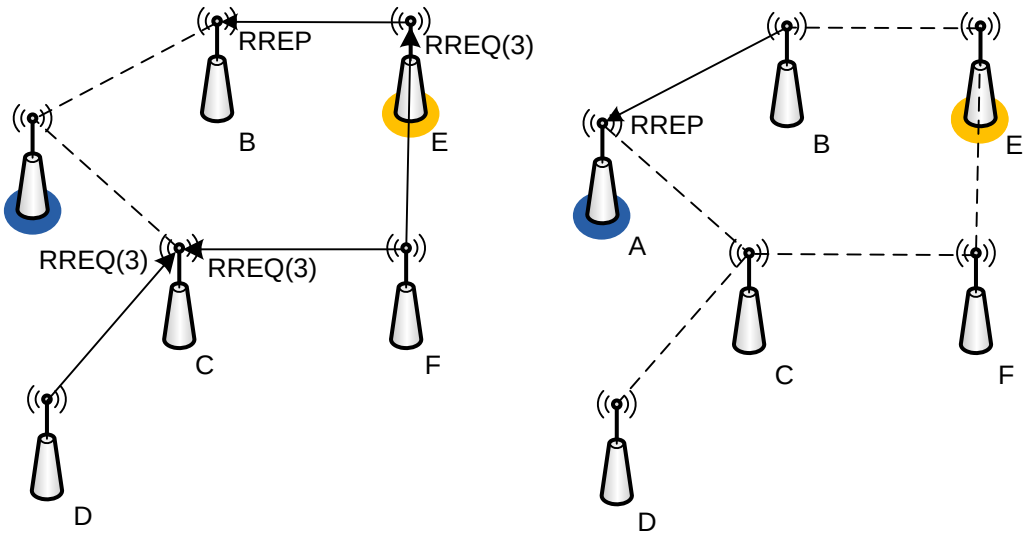
Figure 2.7 illustrates the phases of a route discovery process. To initiate a route discovery, the originator node A floods a route request message (RREQ) to the network. The message has header fields for the originating address A, the destination address E and the number of hops back to the originator (marked in parenthesis in Figure 2.7(a)). If the receiving node, first node B, is not the destination, the discovery process continues. Intermediators store the address of the node they received the request from and rebroadcast it with incremented hop count as shown in Figure 2.7(b).

Eventually the request arrives at the destination node which then unicasts a route reply (RREP) back towards the originator as in Figure 2.7(c). The intermediator node(s), node B in Figure 2.7(d), forwards the reply using the reversed path established earlier when relaying the request. Route discovery is completed when the



(a) The originator node floods the network with a route request.

(b) The intermediator nodes continue the flooding with increased cost.



(c) The destination node replies using the shortest reversed path to the originator.

(d) The intermediator relays the reply back to the originator.

Figure 2.7: Route discovery in AODV floods the network with the request message. When it reaches the destination, the shortest, reversed path is used to deliver the reply.

reply arrives at the originator and the route is stored in the routing table.

If consecutive requests arrive at the destination, it will only reply a request with a lower number of hops to the originator than already recorded in the routing table. In the previous example, node F will not receive a reply from node E since the proposed route has a greater hop count back to node A.

The route discovery process creates potentially a lot of traffic in the network and, if no route to the destination exists, the communication has only wasted precious

energy. A common method of lowering the impact of route discovery packets is to limit the propagation range of the message. A time-to-live (TTL) field is introduced into the packet header and each receiving node decrements the TTL value by one. When the value reaches zero, the packet is no longer forwarded.

When a broken link to a neighbour is detected, all destinations behind the neighbour become unreachable. The detecting intermediator node sends a route error message (RRER) to the originator who reinitiates a new route request for the destination, if the route is still needed. It is also possible for the intermediator to attempt a local repair by initiating a route request of its own.

2.4.5 ISO 18000-7

The ISO 18000-7 standard defines an air interface for Radio Frequency IDentification (RFID) devices operating on the 433 MHz frequency band. The standard covers the communication protocol on physical and data link layer.

All communication is of a master-slave type that is only a master can initialize a communication between devices. The master issues a command to the slaves which then perform the task determined by the command code and reply the master with the result if the task yields any The master can address the packet to either all slaves in range or a single slave.

In RFID a master device is referred as an interrogator and a slave device a tag. Each tag is identified by a unique Tag ID and each interrogator by an Interrogator ID. Devices can be arbitrarily grouped by assigning an Owner ID to devices in the same group.

The medium access is contention-based. The interrogator begins the communication by broadcasting a wakeup signal for minimum of 2.5 seconds up to 2.7 seconds. All the tags periodically listen to the channel to detect the signal. Upon detecting the signal, tags continue to listen to the channel for the command message follows the wakeup signal. After the command message, all the addressed tags will reply the interrogator with the requested data. The reply has an acknowledge flag that indicates whether the tag acknowledged (ACK) or did not acknowledge (NACK) the command. The interrogator closes the communication by sending a sleep command to the tags.

The interrogator ensures the message delivery and validity in all cases. Collisions are handled using a collision arbitration mechanism. All the tag replies must take place within a time window specified by the interrogator. The time window is divided into reply slots and each tag will randomly choose one. A tag is permitted to transmit only during its selected slot. During the reply time window, the interrogator listens to the channel recording both collisions and successful replies. A sleep command is sent to tags whose response was received and they will not participate in the

subsequent collection rounds.

If the interrogator detected collisions, the collection is then repeated for the remaining tags. The procedure is repeated until all tag replies have been received or three collection rounds have passed. The minimum size for the reply time window is 57,3 ms which must be used on the first interrogation round. The on subsequent rounds the interrogator may choose to adjust the window to decrease the probability of collisions.

2.4.6 DASH7

The DASH7 Alliance protocol (D7A) [20] has later described a full protocol stack for RFID tags based on the ISO 18000-7 standard. Unlike traditional RFID protocols, D7A supports tag-to-tag communication. The 433 MHz frequency band enables up to 1 km communication range and higher obstacle penetration than 2.4 GHz band. The maximum data rate is 200 kbit/s.

Typical RFID communication is master-slave type where data is retrieved from the slaves with a pull operation. In a pull the interrogator sends a requests and the receiving tags reply with the requested data. D7A supports also push operations where tags can initiate a data transfer to the gateway. This is useful for example, if tags send measurement data at a constant interval.

2.5 TUTWSN

The WSN research group at Tampere University of Technology (TUT) research group has dedicated over a decade of research work on WSNs. Based on their experiences a complete WSN architecture, TUTWSN, was designed and prototyped successfully [15]. TUTWSN targets at ultra low/power consumption, high configurability and scalability.

TUTWSN MAC protocol utilizes a hybrid version of the clustered topology and tree topology. While clustering provides high connectivity, the cost is increased demand of resources, most importantly energy and memory. Tree topologies have proven more energy efficient in data routing, but lack the robustness of clustered topology against node or link failure due to low connectivity [15]. A hybrid design called multicluster-tree topology targets for the benefits of both domains with acceptable cost. The cluster heads, called headnodes in TUTWSN, connect to a few neighbouring head nodes. resulting in several, partially overlapping tree topologies. The leaf nodes, called subnodes, communicate only with the head nodes.

The access cycle is divided into a superframe and idle time. A head node manages the superframe to which the subnodes and associated headnodes synchronize. The superframe structure is similar to IEEE 802.15.4 with three partitions: a beacon

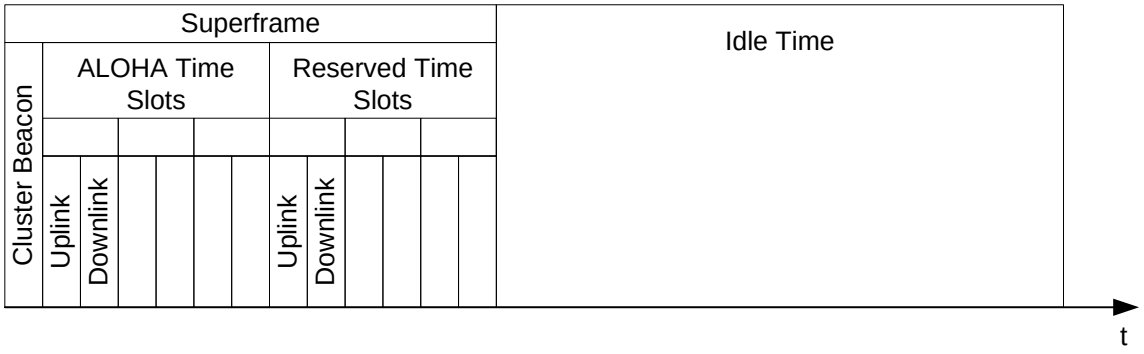


Figure 2.8: TUTWSN MAC access cycle and superframe structure.

slot, contention slots and contention-free slots. The complete superframe structure is shown in Figure 2.8.

During the contention slots, subnodes can request reservation of contention-free slots and association with the headnode [15]. Unlike in IEEE 802.15.4, the chosen protocol is slotted ALOHA [21]. The contention-free slots are reserved for data transmissions. All nodes in the cluster synchronize to the slot boundaries. Each slot is divided into an uplink and a downlink subslot. The receiving node has to wake up to listen only at the beginning of the uplink slot. If no transmission is detected, it may resume the sleep state until the beginning of the next slot. Compared to IEEE 802.15.4, where the coordinator has to listen to the whole CAP, this approach enables the headnode to conserve energy by spending more time in the sleep state. Therefore even the headnodes can be battery-operated.

The TUTWSN routing protocol, TUTWSNR, is based on Directed Diffusion [22] where the sink is the central node and all other nodes keep a record of a gradient which points in the direction of the sink [15]. The gradient determines the next hop of a packet and is redirected whenever a neighbour advertises a lower cost than already known. In TUTWSNR a headnode can keep a record of several routes towards the sink with associated costs.

In network setup phase the sink floods the network with a route advertisement and all nodes adjust the gradient accordingly. The sink advertises its interest for measurement data with an interest advertisement message which is propagated throughout the network. The measurement data propagates towards the sink using the previously established gradients. Later nodes can query the route to the sink or active interest requests from the neighbour nodes. This keeps the network connectivity maintenance overhead low when new nodes join the network or existing links break unexpectedly.

Chapter Conclusion

The general characteristics of WSNs have been discussed in this chapter to give lay a foundation for the simulation models. WSNs have been applied to numerous different applications with highly diverse requirements. Inevitably no single design is applicable to every application. WSNs design is trade-offs between performance, power-consumption, cost and applicability to multiple applications. Both the node hardware and chosen protocols constitute to these factors. International standards have been released to ease the integration of solutions from different vendors.

3. WIRELESS SENSOR NETWORK SIMULATION

The objective of all simulation tools is to provide valuable insight into the modeled phenomenon. In WSN this practically means fast exploration of design space and early feasibility evaluation for new algorithms. Simulation tools can help decide whether a prototype should be built. Simulators focus on different features of the model and therefore choosing the right tool is important for obtaining reliable results.

This chapter discusses the basic principles of simulation focusing on the aspects important in WSN simulation. At the end of the chapter three network simulators are introduced to provide an overview of current simulation options.

3.1 Modeling and Simulation

A model describes the behaviour of a real world phenomenon or a theoretical system. The description is either a mathematical model or a symbolic model, for example a computer program. Modeling and simulation has three phases [3].

1. Defining the model.
2. Running experiments on the model.
3. Analyzing the simulation results.

The first phase is capturing the defining features of the phenomenon. A set of state variables are chosen to represent the state of the phenomenon. This phase abstracts the target, introduces assumptions and simplifies or omits details. The accuracy and the complexity of the model is a direct consequence of these abstraction decisions. The model is then further developed into a computer simulation or design.

The second step is simulation, executing the model on computer by introducing the aspect of time [3]. As time advances, the changes in the state variables reflect the system's behaviour. If the phenomenon behaviour is very slow or very fast, observations are possible by slowing or accelerating time in simulation. Computer-driven simulation enables fast repeated experimentation on the model. By changing the simulation parameters, different scenarios can be created and evaluated.

By observing the model behaviour and output in simulation, the real world equivalent can be estimated. Simulation lets the designer explore possibilities, diagnose problems, indentify constraints and specify requirements. This analysis is the final

step, but may be followed by a new iteration where the model is further defined based on the observations.

Simulation processes time either as a continuum or as discrete [3]. In continuous simulation state variables change continuously with respect to time independently of events in the system. In computer simulation this is imitated by advancing simulation time in small constant-sized steps. The smaller the step, the more accurately time is modeled. Inevitably the simulation runtime increases as the size of the time step decreases. Example applications for continuous time simulation are mechanical and electrical component simulations.

In discrete simulation time advances in respect of events in the system. An event indicates a point of time where state variables may undergo a change. All other time steps can be omitted from the simulation and simulation time advances discretely from event to another. The reduced amount of timesteps makes the simulation more efficient. All simulation tools for WSN simulation are discrete event simulators.

3.2 Discrete Event Simulation

In discrete event simulation the state variables change in zero time. Any change can only take place at a discrete time instance indicated by an event. A simple server-client example in Figure 3.1 demonstrates the concept of events. A client sends a request to the server which processes the request and sends the requested information back to the client. The processing at the server takes a non-zero time.

The initial event, denoted e_1, activates the client to send the request. The reception of the request at the server is indicated with event e_2. It must be followed by a reply to the client so the server model creates a new event e_3 that takes place at the time instant at which the reply must be sent. The server state is changed from *idle* to *processing* and it remains unchanged until e_3 occurs. Therefore the simulation time may jump directly from t_2 to t_3 without missing any details at the server. When event e_3 occurs, the reply is sent. Finally event e_4 closes the communication

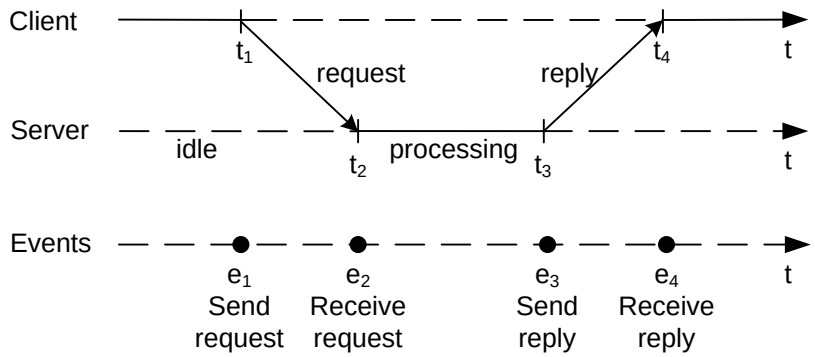


Figure 3.1: Communication between a client and a server modeled with four events.

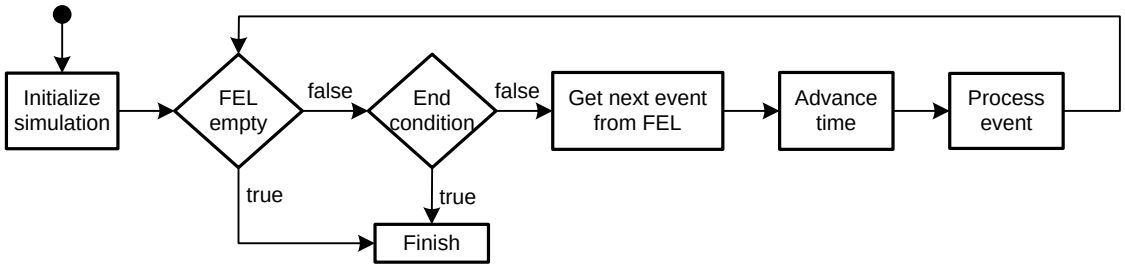


Figure 3.2: Simulation event loop in DES. Adapted from [23].

as the client receives the reply. Events not only indicate a change in the state, but may also trigger activities in the model or create new events.

Every event has a timestamp that indicates the time the event is scheduled to occur [3, 23]. When an event is created, it is placed in the Future Event List (FEL) according to its timestamp value. Events are extracted from the FEL in time order which ensures that later events cannot have impact on earlier events. Events can however cause future events to be added or removed as e_3 was created in response to e_2.

A simulator executes events sequentially by following the algorithm presented in Figure 3.2. In the initialization phase one or more events are created to start the system. Then the first event is taken from the FEL, simulation time is set to the timestamp value of the event and the event is processed. Then the same procedure is executed on the next event. The simulation ends when there are no more events in FEL or a specific end condition has been met. In most cases the FEL will never become empty and the simulation is terminated either by the end condition or user action. Some simulations run only to explore the phenomenon and resolve the end condition. At the end of the simulation there is usually a clean-up phase at which the model instances are destructed and the output records written.

3.3 Requirements for WSN Simulator

Reliable simulation results can be achieved only through accurate modeling. Any simulator for WSNs must provide means to model the node platform, different communication protocols and the physical environment [4]. The simulator must obviously have tools for collecting and analyzing the simulation output and statistics.

WSN simulation has two important aspects: the communication and the node platform [15]. The communication protocols, the wireless medium, the transceiver unit and the application all constitute to the communication perspective. The memory, computation and energy constraints relate to the platform of a single node device. Platform model should be able to simulate the data processing using the intended algorithms. To accurately simulate both aspects, a separate simulator is

often needed for the network-wide communication and the node hardware platform. However, the aspects are not decoupled and a joined simulation of the two is preferable to two separate simulations.

Scalability is currently one of the main concerns. The simulation should execute fast even if the network consists of hundreds of thousands of nodes. Component-based architecture has been reported to scale better than object-oriented [24]. However, object-oriented approach makes recomposing the models easier which is often desirable. For example, layers of the protocol stack may be replaced with others while exploring options for optimal communication for a specific application.

An increasingly important feature is the visualization of the model, collected data and sensor behaviour [3, 4]. Graphs, images and animations are all techniques used in visualization. The objective is to help understand the model and the information obtained through simulation. This is especially important for large quantities of data and information that varies over time.

3.4 Network Simulators

There are a lot of simulator tools available for wired and wireless networks. J-Sim [24], GloMoSim [25], NS-2/NS-3 [26, 27], SENS [28], and SensorSim [29] are all non-commercial simulators, just to name a few. Commercial simulators include OPNET Modeler [30] by Riverbed, Simulink [31] by MathWorks and NetSim [32] by Cisco. The following sections introduce three well-known non-commercial simulators along with their pros, cons and central features. It helps to understand how the requirements discussed in the previous section are currently accounted for. OMNeT++ is discussed in detail in Chapter 4.

3.4.1 J-Sim

J-Sim [24] is an open source WSN simulation framework written in Java [33]. It utilizes a component-based architecture that scales better than the object-oriented approach used in many simulators. Real hardware sensors can be connected to the simulation for authentic data from outside the network. Furthermore, real applications can be ported on top of the protocol stack. A script interface allows integration with scripting languages such as Perl [34], Tcl [35] and Python [36].

J-Sim is more scalable than object-oriented simulators such as NS-2, but complicated to use [5]. While other issues do exist, the greatest drawback is that only the IEEE 802.11 MAC protocol is supported. While Java makes the framework platform-independent, extensible and reusable, it also imposes some of its problems to the framework itself. Simulation is less efficient than in frameworks based on other languages that must be compiled to the native language of the platform. The

latest update for J-Sim was released in 2006.

3.4.2 GloMoSim

Global Mobile Information System Simulator (GloMoSim) [25] is a parallel discrete event simulation library. It is designed for efficient simulation of wireless multi-hop ad-hoc networks. The simulation library is implemented using C-based programming language called PARallel Simulation Environment for Complex Systems (PARSEC) [37]. PARSEC Visual Environment (PAVE) accounts for visual design and configuration of simulation models.

GloMoSim provides a set of library modules each simulating a specific protocol in the protocol stack. Strictly defined Application Programming Interfaces (APIs) have been designed for each layer. The API defines how the layer communicates with the neighbouring layers. Any layer can be replaced with any other that obeys the same API.

The simulation consists of physical processes (entities) and time-stamped messages representing events. Traditional approach suggests that each node in the network is represented as an entity associated with other entities representing its protocol layers. However, parallel simulation does not scale well with the number of entities making this approach unsuitable for very large networks. In GloMoSim scalability has been accounted for by dividing the network into partitions. An entity represents a single protocol layer for all the nodes in the partition. The chosen approach significantly reduces the amount of simulation entities.

GloMoSim has proven as an effective tool for IP network simulations, but it is not capable of simulating any other type of networks [5]. Another major restriction is that all events must be generated by the nodes in the network. Due to these limitations, many WSNs cannot be accurately simulated. The original GloMoSim environment was released as open source, but it has not been updated since 2000. Instead, it has been replaced by QualNet, which is a commercial simulation tool built on GloMoSim..

3.4.3 NS-2

One of the most popular open source network simulators has been NS-2 [26]. It is an object oriented simulation tool targeted for modularized, extensionable network simulation. Simulations are implemented using C++ for protocol models and OTcl (Object oriented extension of Tcl) for controlling the simulation environment. This makes the simulation entities run fast but changes are slow since they require recompilation. OTcl runs slower, but changes are fast. Due to its popularity it is not surprising that numerous contributors have released public protocols libraries.

NS-2 has several known issues [5]. Due to its object-oriented approach, it does not scale very well with the network size. It also imposes restrictions to customization for packet formats, energy models and MAC protocols, for example. Furthermore, it completely lacks the application model. Finally, although the use of two different languages is justified, it makes the simulator more difficult to use. The user has to learn two languages and how they are connected in the simulation environment.

Many of these issues were addressed in the succeeding NS-3 simulator [27]. The simulation core is C++ while Python has replaced OTcl. Much like NS-2, NS-3 focuses on Internet protocols, but is not limited to them. NS-3 also provides interaction with real world and packets can be sent and received in the simulation through network interfaces.

Chapter Conclusion

Although simulation is widely adopted in many fields, the characteristics of WSNs have proven difficult to capture. Modeling a complex network of devices requires not only detailed models but also efficient simulation environments. A WSN simulator must be scalable, modular and extensible. There are a lot of simulation tools available, yet choosing one for specific use is not a trivial task.

4. OMNET++ SIMULATION FRAMEWORK

This chapter describes the OMNeT++ simulation framework. It presents the simulation tool and introduces the modeling concepts. The concepts define how new models can be implemented for simulation.

4.1 Overview

Objective Modular Network Testbed in C++ (OMNeT++) is a component-based discrete event simulator for wired and wireless networks. The simulation models are implemented in C++ and attached to the environment using NEtwork Description language (NED). OMNeT++ is free for educational use, but has also a commercial lisence available. The commercial version is known as OMNEST.

OMNeT++ is distributed as a zip-archive file. After the archive has been extracted, the source files are compiled for the target platform. Once the compilation has finished, the simulation environment is ready for use. The archive contains only the simulation environment, no model libraries are included. A step-by-step installation guide is provided in [6].

The environment includes an Eclipse-based [38] Integrated Design Environment (IDE) for development, simulation and project management. An example of the IDE view is shown in Figure 4.1. The directory structure of the open project is shown on the left. The modeled networks and modules can be assembled and connected in a graphical editor shown in the middle. By selecting the Source view, the editing can be done in a text format. All changes will be visible in both views. The text editor is also used for C++ development for implementing the module behaviour. The IDE provides tools also for compilation, debugging and simulation run.

The simulation can be run on either of the two user interfaces: Cmdenv or Tkenv [23]. Cmdenv runs the simulation from command line and may utilize scripts for repeated batch execution. Tkenv provides a graphical representation of the simulation run. An example is shown in Figure 4.2. A red rectangle is used to identify the currently active module. Transmitted messages are denoted with colored circles whose color depends on the message type.

Tkenv shows the visualization and the run-time output in a separate window. An example of the run-time events is shown in Figure 4.3. The simulation execution and speed can be manipulated using the controls in the toolbar at the top. Simulation

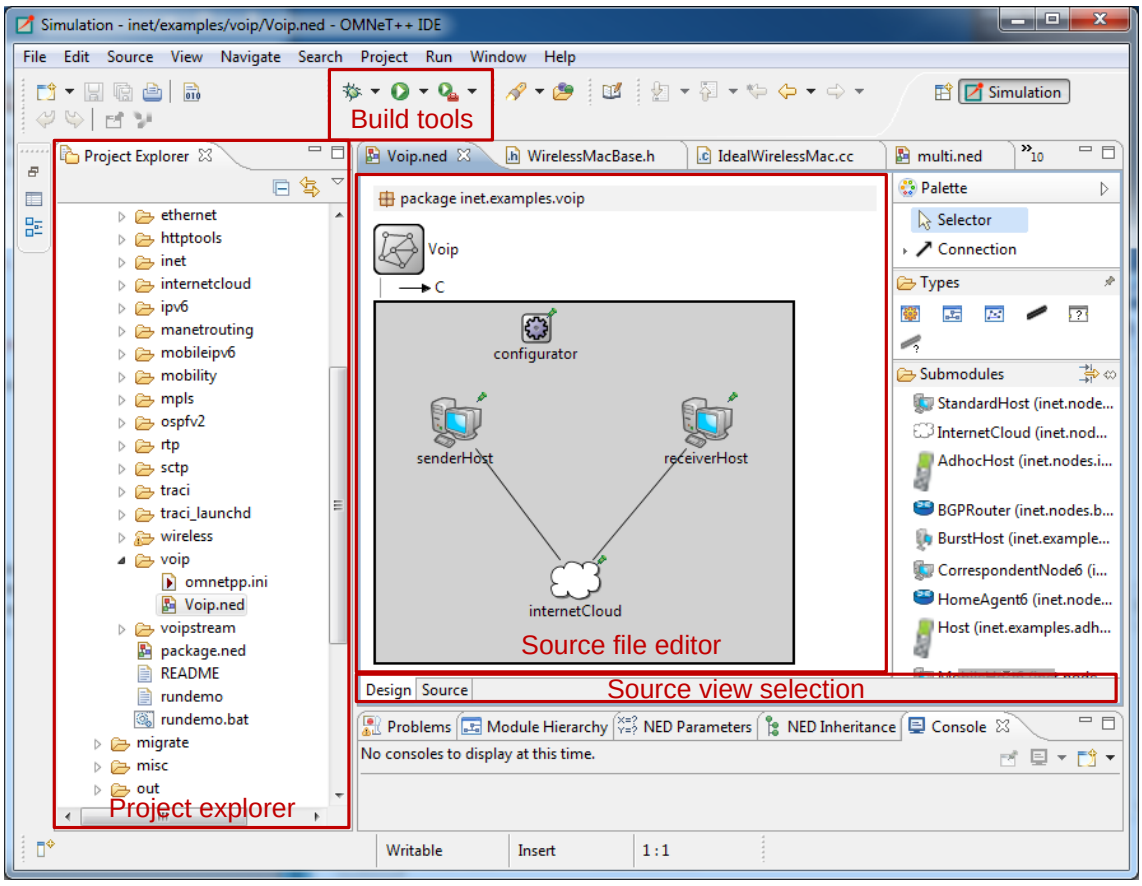


Figure 4.1: OMNeT++ IDE is build on Eclipse.

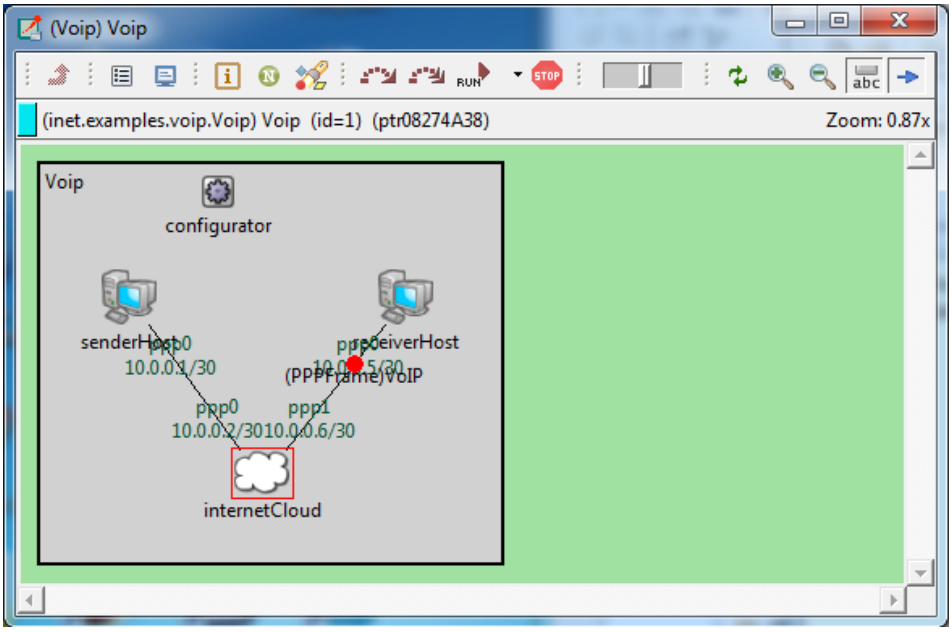


Figure 4.2: Tkenv provides a clear visualization of the network.

can be run for example one step (event) at a time, in normal speed or in express speed. The future event list is shown on the left. All the events and the module

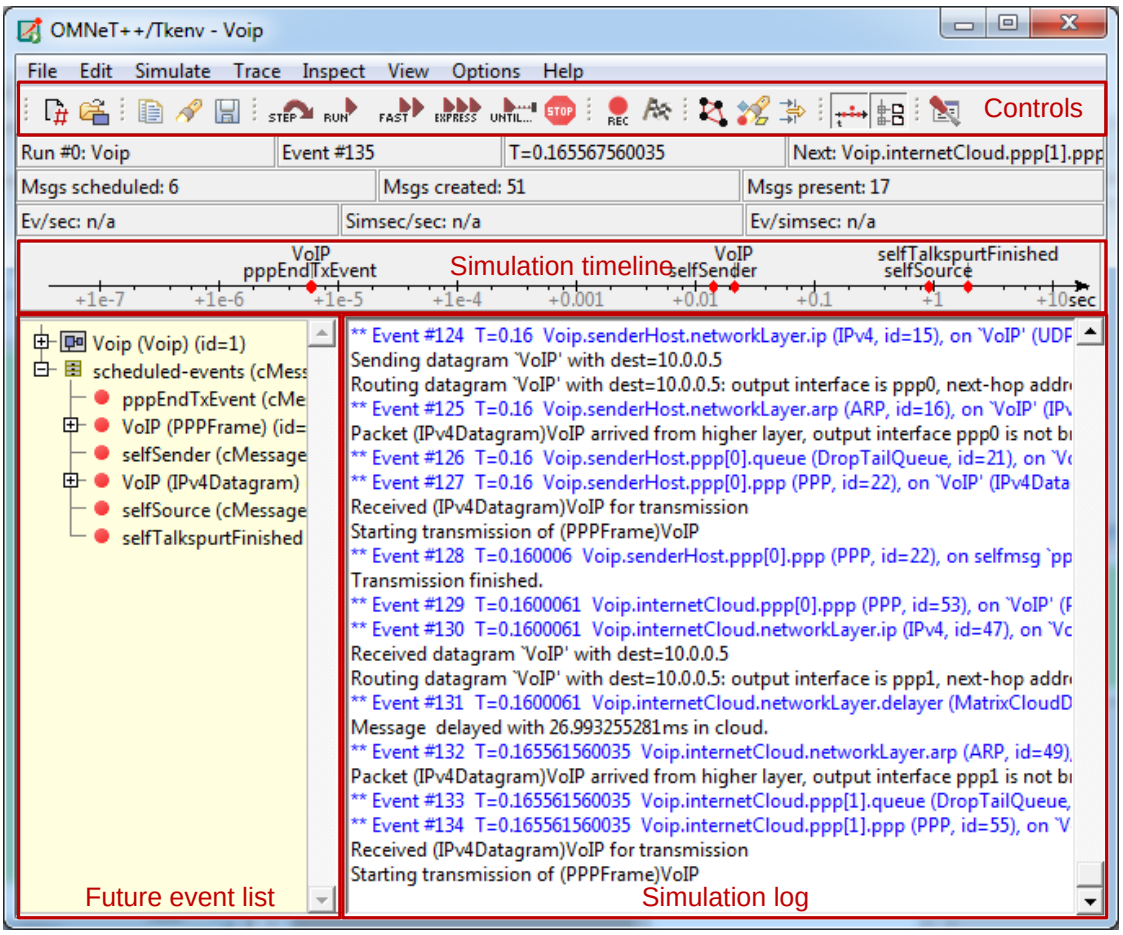


Figure 4.3: The simulation details are shown in a separate window.

outputs are printed in the simulation log.

The IDE has also analysis tools for the simulation results. The results can be browsed, processed and visualized as charts and plots. The Scave Tool provides a lot of the same information on command-line and can be used to produce data files for other tools for further processing.

4.2 Modeling Networks

The following subsections describe the modeling concepts in OMNeT++. These concepts must be followed in all simulation models. Creating new models requires understanding these underlying structures.

4.2.1 Messages

In OMNeT++ instances of class *cMessage* represent events [23]. They encapsulate all forms of communication in the system: from a simple signal that activates the system to arbitarily long data packets transmitted from a device to another. The arrival of a message clearly indicates a possible change in the receiver's state. By

sending the message to itself, an entity can activate after a pre-determined time. Such messages are known as self-timers. In the server-client example e_2 represents a packet from the client to the server and e_3 a self-timer. Since messages in general often introduce a timestamp field for e.g. synchronization purposes, OMNeT++ uses field *arrival time* to maintain time order in FEL.

Lower layer in protocol stack often requires additional information from the upper layer for processing a message. Additional control information can be attached to a cMessage with method *setControlInfo()*. The lower layer can then retrive the control info object by using *getControlInfo()*.

cMessage has a subclass cPacket for modeling network packets, for example Ethernet frames. Packets, unlike messages, may encapsulate other messages and have a field for the bit length of the packet. However, these features alone are not enough to capture all the information packet headers contain. Writing an exteded subclass with additional data members is often trivial, yet laborous. To address these two problems, OMNeT++ provides a mechanism for generating classes from a NED message definition. The generated class may introduce new fields e.g. addresses and inherit other definitions. The generated subclass has private variables along with setter and getter methods for all the defined fields. An example definition of an Ethernet frame is shown in Listing 4.1. Some of the fields are omitted for clarity.

```
packet EthernetFrame
{
    bitLength = 208;        //Header length
    int destAddr;           //Destination MAC address
    int srdAddr;            //Source MAC address
    int crc;                //Frame Checksum
}
```

Listing 4.1: Message definition of an Ethernet frame.

The definition may also set initial values or alter inherited fields as bit length is set in the example. Comments can be inserted as in C++. Using message definitions considerably improves re-use of existing message classes and reduces the effort of creating new message types.

4.2.2 Modules and Channels

Modules represent the entities in which events take place and new events are generated in. They model for example actors, devices, protocol layers, user inputs and any other components composing the simulation target. OMNeT++ uses two types of modules: simple modules and compound modules.

Simple modules are the active parts of the simulation. At the beginning of the simulation, before the first event takes place, *initialize()* is called for each module

for internal initialization. When an event is scheduled to occur, the simulation environment passes it to the receiving module for processing. The module reacts to the event as described the *handleMessage()* function. When the function returns, next event is extracted from the FEL. Function activity() can be used instead of handleMessage in process-style models. At the end of the simulation, the simulation environment calls *finalize()* for each module for output recording and clean-up.

A compound module has no behaviour of its own. It instead contains an arbitary combination of other modules and interconnections. This enables hierarchical simulation structure and model design. The top-level component of the simulation must be tagged with the *@network* property. The connections are implemented using specialized modules called channels.

NED is used to declare modules, interfaces and to assemble compound modules from submodules and channels. Both simple and compound modules are parametrizable. The C++ implementation can access the NED parameters at runtime and therefore a change of parameters does not require recompilation. NED uses a lot of metadata annotations for labeling and attaching properties to objects. An example simple module description for a client is shown in Listing 4.2.

```
simple Client like IClient
{
    parameters:
        @class(SimpleClient);
        double send_interval @unit(s) = uniform(0s,1s);

    gates:
        output toServer @label(EthernetFrame);
        input fromServer;
}
```

Listing 4.2: Example NED definition of a simple module

The client module inherits a common interface, *IClient*, as indicated by the *like* keyword. The first parameter demonstrates the use of metadata properties: *@class* defines the C++ class that implements the functionality. The second parameter is used to control the behaviour by assigning a uniform distribution that defines a pseudo-random time between two consecutive messages to the server. It also has a metadata property that defines seconds as the expected unit.

The gates-section lists the gates channels can be attached to. The gate must have one of the three directions: input, output or inout. A gate may also be declared as an array. This is useful for connecting several modules whose messages are handled in a similar way. For example, a server may have an array for several clients to connect to. The gates are expected to be connected, unless they have the *@loose* or the*@radioIn* label. A gate can have a metadata label to inform of the expected

message type. In Listing 4.2 the arriving messages from server are expected to be of type EthernetFrame.

A channel creates a unidirectional connection by associating with exactly one output gate and one input gate in the connected modules. When a module sends a message to an output gate, the channel delivers it to the input gate of the other module. Bidirectional channels are impelented by attaching two channels together. OMNeT++ features three default channels: ideal channels, delay channels and datarate channels. An ideal channel delivers messages in zero time from one component to another without altering the message. A delay channel delivers a message to the recepient after a predefined delay. A datarate channel calculates the delivery time for every message from the channel's set datarate and the packet's bit length. It may also introduce a bit error or a packet error to the message. For most applications these three channels are sufficient, but it is also possible to implement custom channels. Like simple modules, their behavior must be explicitly described in C++.

OMNeT++ features a signaling mechanism. Signals can be used for example to notify objects or simulation environmnent of changes in the model, gather statistics and implement communication channels where the subscribers do not know about the sender(s). Any component (module or channel) can emit signals using the *emit()* function. A data value can be associated with the signal. The emitted signal propagates up in the module hierarchy until it reaches the system level. A listener, an object inherited from the cIListener class, is an entity that receives signals. To receive signals the listener subscribes to desired signal(s). When a signal is emitted, the subscribing listener is notified and its *receiveSignal()* function is called. It is possible for the listener to figure out the source of the signal, but it is often desireable to conceal the source. For example, in the server-client example, assuming that the client and the server emit signal *packetDrop* on a failed transmission, a listener on system level aggregates the number of dropped packets in the whole network.

4.3 Simulation Output

The simulation statistics and output can be generated by either directly as a part of the behavior in C++ code or by using the signal mechanism [23]. Traditionally modules keep record of their own statistics and write the statistics at the end of the simulation when *finalize()* is called. The drawback with this method is that any changes requires changes in the code and thus recompilation.

The output record uses scalars and vectors. A scalar is a summarizing value, for example the number of sent packets by a module. Vectors are collections of time-value pairs and can be used for example to record the end-to-end delay of packets. For more complex statistics, such as standard deviation and histograms, OMNeT++ provides own estimation classes that can be used inside components for

output recording.

NED has property *@statistic* for declaring signal-based statistics in a component. The input signal(s), processing, what properties and the output record(s) must be declared. Listing 4.3 shows an example use of a statistic declaration in the previous client example. The statistics trigger when signal retry is emitted. If no source signal is specified, the signal with the name of the statistics, connectionRetry, is considered the input. After the signal has been emitted, all the given records, count and last, are updated. The total number of connection retries are stored in the count record and the last value of the signal in the last record. OMNeT++ supports result filtering with other common aggregate functions such maximum, minimum and mean value. Vector recording the values is marked optional using the question mark. Other optional element include for example the unit for the values.

```
simple Client
{
    parameters:
        @class(SimpleClient);
        @statistic[connectionRetry](source=retry;
                record=count,last,vector?);
        double send_interval @unit(s) = uniform(0s,1s);

    gates:
        output toServer @label(EthernetFrame);
        input fromServer;
}
```

Listing 4.3: Statistic recording using signals.

Finally, watches can be used to monitor component's variables at simulation run-time. Watches must be defined inside the component implementation using the WATCH() macro. The watched variable values can also be changed in the simulation environment. Finally, watches are used for creating snapshots of the simulation. The snapshot() function writes all the values monitored by watches into a snapshot file. Regular snapshots during simulation enables monitoring the model behavior over time.

4.4 Related Work

The OMNeT++ simulator has been utilized in WSN simulation also earlier. This section introduces some of the most notable work. Like the simulator itser, most of them are not yet finished.

INET Framework [39] is a simulation library of protocol models for OMNeT++. It includes various models for all protocol layers and supporting modules e.g. routing tables, device mobility, wireless channel model, propagation models and simulation

Table 4.1: Protocols provided with INET Framework [39].

Protocol layer	Protocols
Application	HTTP, FTP, DHCP, Video streaming, VoIP
Transport	TCP, UDP, SCTP, RTP, RTCP
Network	IPv4, ICMPv4, ARP, IGMPv2, IPv6, ICMPv6, MIPv6, HIP
Routing	LSR, OSPF, BGP, RIP, RSTP, AODV, DYMO, DSDV, SDR, OLSR
Data link	Ethernet, PPP, IEEE 802.11, IEEE 802.15.4, IEEE 802.16

scenario manager. The available protocols and models, excluding those reported incomplete or unstable, are listed in Table 4.1. The latest stable version of INET was released in August 2013. The INET Framework has been used as a part of this work.

The lower layer implementations lack a lot of detail and provide only very abstract behavior. Other projects and branches of the INET have been initiated to provide more accurate models. Some notable projects are MiXiM [40] for radio and MAC implementations, INETMANET [41] for ad-hoc networks and OverSim [42] for overlay and peer-to-peer networks. While most of the provided models are easily utilizable and mature, the documentation of various models is incomplete and outdated. The documentation has not been updated since June 2012.

Castalia [43] is an OMNeT++ based simulator for networks of low-power wireless devices such as WSNs and Body Area Networks (BANs). The simulator focus is strongly on the communication perspective. Wireless channel transmissions are modeled in great detail based on empirically measured data and are easily configurable for different scenarios. Several MAC protocols, such as IEEE 802.15.15 and T-MAC [44], are also provided. While routing is less important to communication, a basic routing protocol, called multipathRings, is also provided for convenience. Other notable features include monitoring of node energy consumption and clock drift, mobility, sensor device simulation and extensibility. The latest version was releases in March 2011.

The Castalia simulation structure is shown in Figure 4.4 [43]. Wireless channel module simulates the data physical transfers between the nodes. Each node consists of a protocol stack, a resource manager, a mobility manager and a sensors manager. The resource manager monitors the node resource state and usage. The monitored resources include energy, cpu, memory and clock drift. Mobility manager keeps track and distributes of the node position information and handles the node movement. Sensors manager encapsulates all the sensing devices and their functionality. Monitored phenomenon such as temperature and illumination are modeled as individual physical processes.

The value of the physical quantities vary in spatial and temporal domains and may

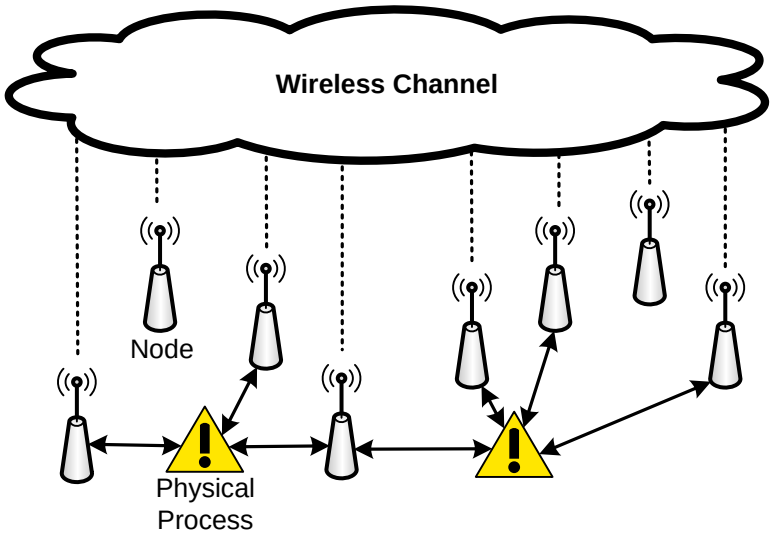


Figure 4.4: Network structure in Castalia.

be affected by several sources. Sensor manager request the value from the physical process which will reply with the value in the corresponding time and place [43]. For example, two nodes monitoring room temperature read different measurement values since they are located differently inside the same room. Furthermore, multiple heat sources e.g. radiators, devices and people cause local changes in the temperature which will over time increase the overall room temperature. For each monitored physical process, there is one sensing device in the node's sensor manager.

Chapter Conclusion

This chapter has discussed the possibilities of OMNeT++ simulation framework. It has several advantages over the other simulators described in Chapter 3: The framework is scalable and protocol independent, unlike J-Sim, the framework itself and the protocol models are highly customizable, unlike NS-2, and new updates become available on a regular basis. OMNeT++ provides a full set of tools for visualizing the simulation run and analyzing the simulation results.

5. WSN SURVEILLANCE SIMULATION

The first simulation case, the WSN surveillance simulation, is based on an existing prototype implemented earlier in a WSN research project at TUT. The prototype is shown in Figure 5.1. It is composed of a TUTWSN node and a WLAN video camera and the node can activate the recording on demand. The surveillance concept is imported to the simulation environment. The results obtained with the prototype could be compared with the simulation. Simulation model also enables experimenting with a larger network.

The network is logically divided into hierarchy levels. In normal operation energy-conscience nodes are used for sensing. They compose the the lowest hierarchy level. A node detecting a triggering event alarms the network and wakes up the higher level which utilizes more efficient, but more energy consuming communication. The objective of the simulation is to study the impact of two radios in terms of energy consumption and performance.

Figure 5.1: A working prototype has a WLAN camera and a low-power TUTWSN node for camera wakeup.

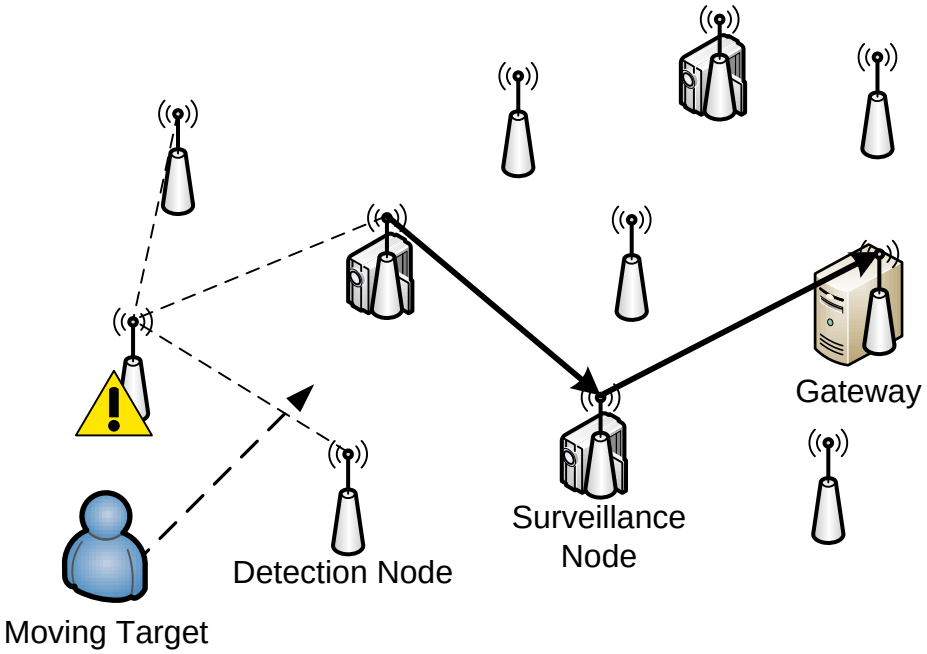


Figure 5.2: Simulation case setup.

Devices on the higher level model the prototype. They are equipped with surveillance cameras and wake up upon receiving the alarm. They utilize high data rate protocol to transfer video stream of the event to a server gateway. The network configuration is shown in Figure 5.2. Once the event has passed, the streaming nodes resume the sleep state to minimize their energy consumption.

5.1 Requirements

The network has essentially three kinds of devices: low-power sensing nodes with motion detectors, camera-equipped nodes capable of low-power and video streaming communication and sink device(s) to receive the video stream. The sensing nodes must be equipped with motion sensors that react to movement in range and trigger the application. The application exploits the underlying protocol stack for alarming other devices.

The nodes with cameras will require two separate protocol stacks: one for low datarate communication with the sensing nodes and another for high datarate video streaming. The node may be equipped with motion sensors or rely on the sensing nodes to inform it of the alarm events. The video streaming protocol stack is initially turned off while the low-power protocol monitors the deployment area.

The server gateway requires only the video streaming protocol. It must be ready to receive an incoming data stream at all times. The radio module must constantly listen to the channel unless the gateway is transmitting. It can be assumed that the gateway device is not battery-operated and idle listening is not an issue.

Finally, a mobile motion stimuli is required to model a moving target in the deployment area. Its sole purpose is to trigger the motion sensors. Without it the network would never wake up.

5.2 Implementation

This section presents the implemented models required in the simulation of the described surveillance application. The most notable models are a low power routing and a MAC protocol that could not be obtained from third party libraries. Their implementation has a major impact on the accuracy of the simulation results.

5.2.1 Devices

The INET Framework implementation of IEEE 802.11 WLAN video streaming was chosen as a starting point for the high data rate protocol stack. The provided radio model could also be used as a base for the low-power radio as well. Other required protocols, that is application, routing and low-power MAC, had to be designed and implemented since no applicaple models are available.

The low-power and high datarate protocol stacks have to be usable together in one device and separately in other devices. Initially, they have to be instantiated in the surveillance nodes as shown in Figure 5.3. Models implemented in this work are denoted with green. Models from other libraries are denoted with blue. Models denoted with yellow originate from other libraries, but have been been modified for this application.

From the communication perspective, the sensing mechanism is the least interesting. Therefore a simple implementation of the motions sensors is sufficient. Every

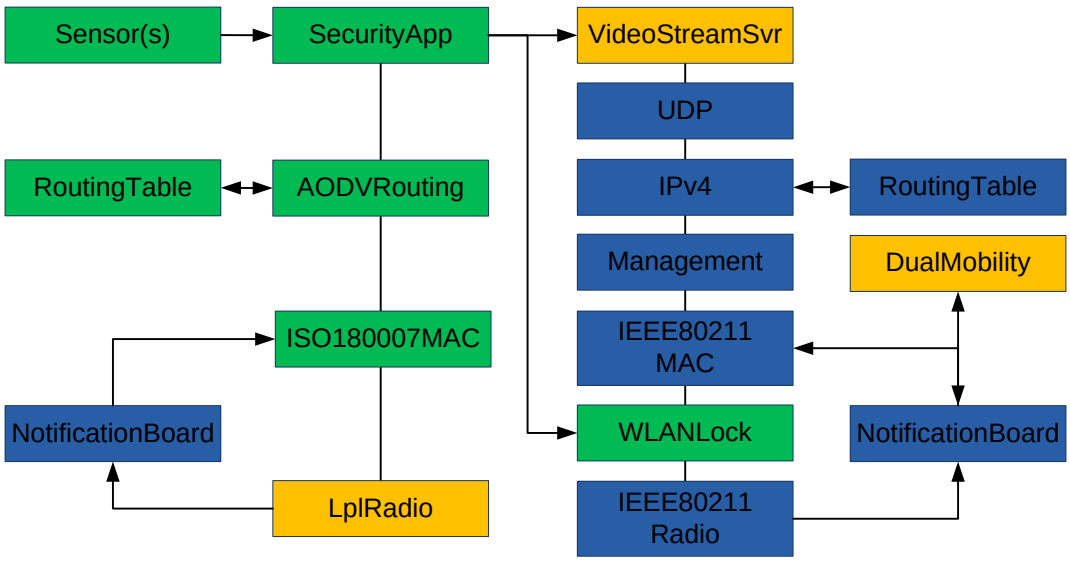


Figure 5.3: The dual protocol stack in surveillance nodes.

time a target module moves, the motion sensors receive a notification with the new coordinates of the target. The sensor then calculates the distance to the stimulus. If the stimulus is within the sensor's range, the sensor sends a reading to the application layer. The sensor is configurable with parameters for sensing range and delay for sending the sensor reading.

The sensor utilizes the node's mobility module to get its location at the beginning of the simulation. The mobility implementation in the INET framework was modified to support two radios in the same device. The existing implementation imposes a constraint that the sensing nodes must remain stationary. In the target scenario this is acceptable.

5.2.2 Surveillance Application

The surveillance application has a simple objective: raise an alarm in the whole network upon detecting an event of interest. Once the application receives a sensor reading, it will immediately send a message to the network informing other nodes of the event. All other nodes receiving the alarm message will forward the alarm message effectively flooding the network. The alerted node will immediately activate the protocol stack for delivering video stream to the gateway.

The INET implementation of WLAN MAC and radio protocols assumes that the radio is always on. To minimize the energy consumption in the nodes the whole stack had to be shut down. Without altering the INET layer implementations, the simplest way to implement this is to add a lock module between the MAC and the radio. The application can unlock the module by sending an unlock message to it. While the module is unlocked, it will let all the messages from the MAC to the radio pass throught. Otherwise it will discard all messages and thus prevents all the communication. The module will lock again when the application sends another message to it or after a configurable idle time if no messages are sent or received.

The surveillance application sends another request to the video application. The original INET implementation was altered by an additional port to the module to receive the start message. Upon reception the application initializes a video stream transfer to the gateway.

5.2.3 Routing

AODV was chosen for a low-power routing protocol. It has been successfully utilized in WSNs earlier and it provides all the required functions for routing in the target scenario. Some features have been omitted for simplicity. This section describes the implemented model in detail.

Two features have been omitted from the model. First, the packets do not have

a TTL field and are allowed to flood the entire network. In networks with relatively small number of devices this will not cause severe performance issues. Second, when a link breaks, the intermediator attempts a local repair by initiating a route request to the destination, but will not send a route error message upstream to the originator. This simplifies the model and lowers network traffic, but may result in unoptimal routes. For example, if the destination node has moved closer to the source, a new route through the intermediator is longer than what could be discovered by the source.

The NED description of the routing layer is shown in Listing 5.1. It inherits a common IRouting interface which allows fast swapping of the routing layer with other implementations. The identifying node id of the device must be given as a parameter. It is interpret as the address of the device in the network. The routing layer has four gates for connecting to the neighbouring layers. An additional gate is provided for future use for connecting to the CPU module. The *@loose* attribute ensures it can be left unconnected.

```
simple AODVRouting like IRouting
{
    parameters:
        @class(AODVRouting);
        @display("i=block/network2;q=downQueue");
        int nodeId;
        // Inherited from IRouting:
        bool routingCapacity = true;
        string routingTableName = default("routingTable");

    gates:
        // Inherited from IRouting:
        input upperIn @labels(AppMsg);
        output upperOut @labels(AppMsg);
        input lowerIn @labels(RoutingMsg);
        output lowerOut @labels(RoutingMsg);
        inout proc @labels(NodePrimitiveMsg) @loose;
}
```

Listing 5.1: AODV routing layer description in NED.

The decision algorithm for messages from the higher layer is shown in Figure 5.4. Any message marked as a broadcast by higher layer can be sent immediately and is passed to the lower layer. For other messages the destination address is read and checked, if an an active route exists in the routing table. If a route exists, the message is forwarded as a unicast to the next node along the route. If no route exists, a route request is initiated and the message is buffered until the discovery process is done.

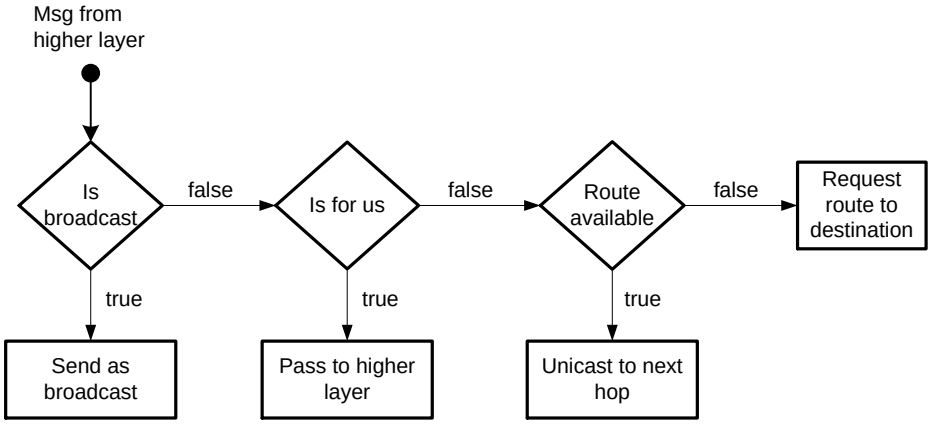


Figure 5.4: Routing algorithm for messages to transmit.

The handling of a received message obtainded from the lower layer is not as straightforward. The decision algorithm is shown in Figure 5.5. First, the address of the previous intermediator is stored in the routing table with a hop count of one. Further actions are based on the message type which is read from the header fields. The handling of route requests and replies is detailed in Figures 5.6 and 5.7 respectively.

Data packets are next checked, if the received packet is a duplicate of an earlier received packet. In AODV the duplicates are identified by the combination of the source address and the sequence number. The model simply uses message tree ids provided by the simulation engine. Regardless of the detection method, duplicate messages are discarded to avoid congestion of the network. All incoming broadcast messages are passed to higher layer and a copy is broadcast to all neighbouring nodes. A unicast message addressed to the receiving node is passed to the higher layer as well. Otherwise, if an active route to the destination is found in the routing table, the message is sent to the next node towards the destination. If no route exists, a route discovery is initiated. Nodes without routing capacity can not forward any, broadcast or unicast, messages, but the rest of the functionality remains the same.

Figure 5.6 shows the decision algorithm for route requests. A route request may be set as a route repair when a broken link is detected and new route request is initiated to discover an alternative route. Receiving a route repair message invalidates all routes in the routing table to and through the destination of the repair message and initiates a route discovery.

Other route requsts are first checked for the hop count back to initiator. If the hop count is higher than the count in the routing table and the request message is seen before, it is discarded. A route with a lower hop count is updated to the routing

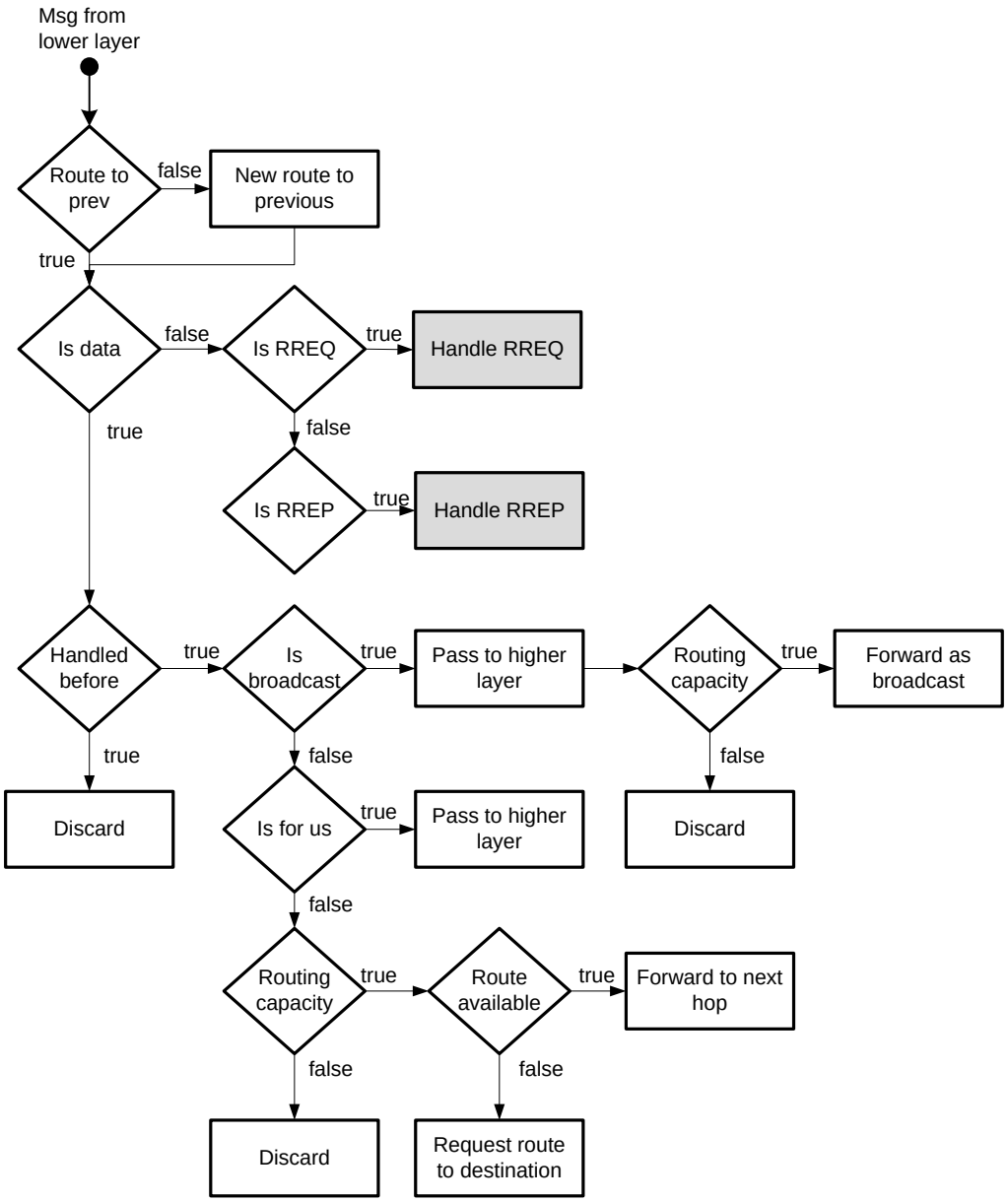


Figure 5.5: Routing algorithm for received messages.

table. If the receiving node is the destination of the message, a route reply is sent back to the originator. Otherwise, an active route to the destination is looked up in the routing table. If a route to destination is already knwon, the request is forwarded as a unicast to avoid unnecessary broadcasts. If there is no route in the routing table, an entry for it is created and the route status is set as *requested*. The request is then forwarded as a broadcast. If another request to the destination arrives later in time while the route status is *requested*, the request message is discarded.

Finally, the routing algorithm for received route replies is shown in Figure 5.7. Any reply for which there is no route in the routing table is immediately discarded. The hop count back to the source of the reply is compared to the known hop count

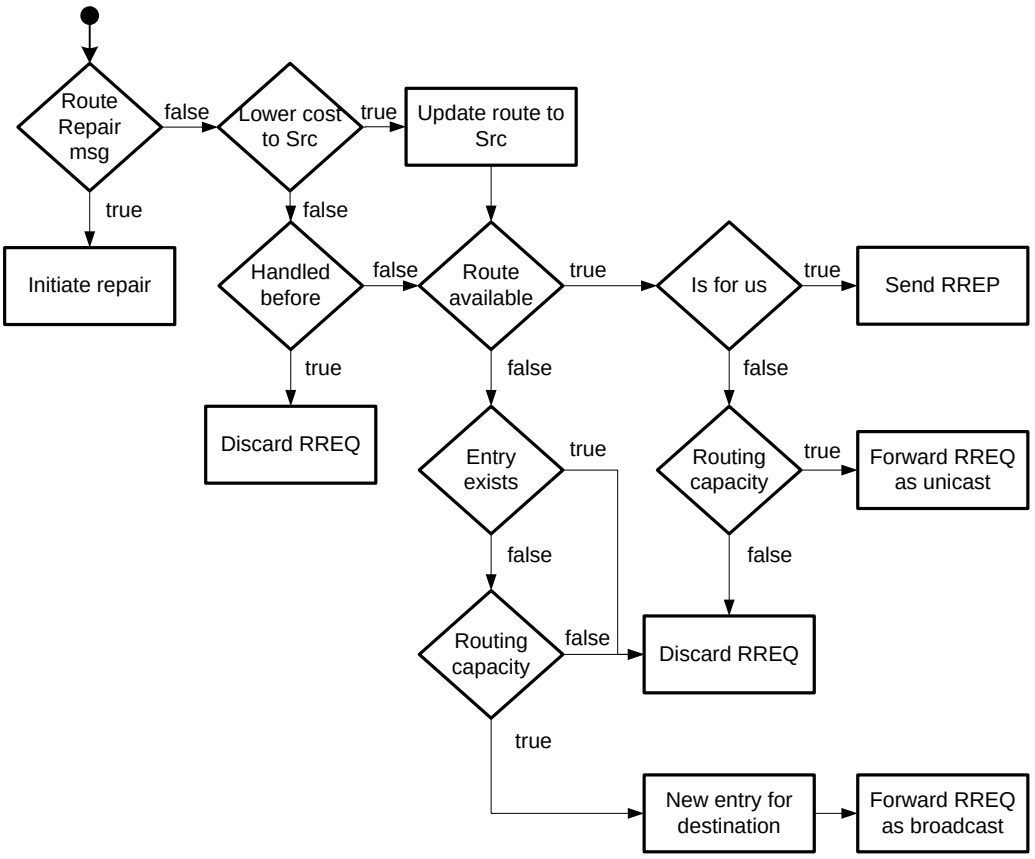


Figure 5.6: Routing algorithm for received route requests.

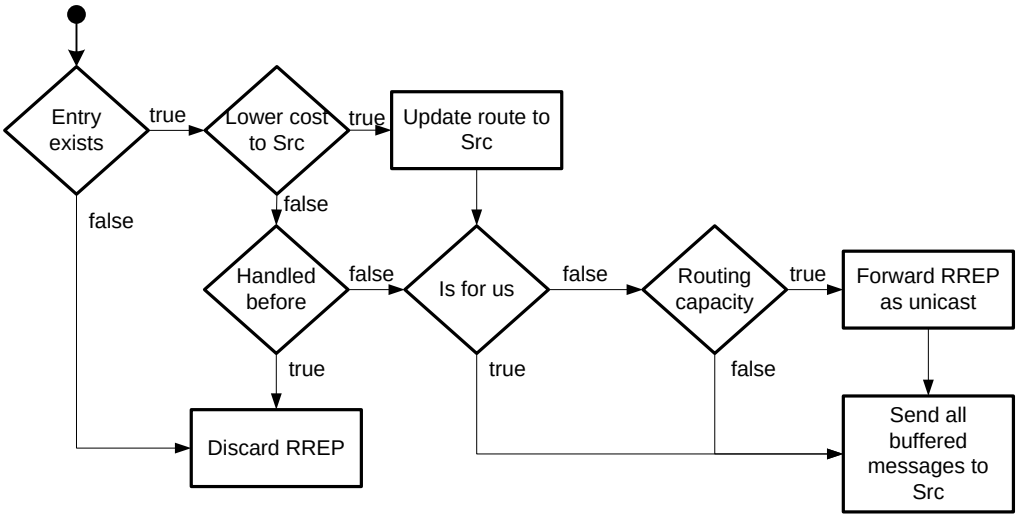


Figure 5.7: Routing algorithm for received route replies.

in the routing table. A shorter route is updated to the routing table. A longer route is discarded. If the reply is for the receiving node, i.e. it is the originator of the route request, all messages in the transmit buffer are sent towards the destination. Otherwise the reply is forwarded towards the originator.

5.2.4 MAC protocol

The simulated MAC protocol, ISO-MAC, was implemented using the ISO-18000-7 standard as a starting point. In the standard each tag is identified by a unique Tag ID and each interrogator by an Interrogator ID. The implementation bundles these to a single node ID. It is used to identify and address the node on both the ISO-MAC and the routing layer. Tags can be arbitarily grouped by using Owner IDs, but this feature is not used in either of the simulation cases.

In order to be usable in as WSN MAC protocol, the standard functionality was extended by allowing a device to function as both an interrogator and a tag. This enables multihop-routing in the network. When a node has data to send, it assumes the role of an interrogator. All nodes periodically listen to the channel and assume the role of a tag upon detecting a wakeup signal from a neighbouring node. To minimize collisions, an interrogator will always listen for any ongoing wakeup signals before initializing a wakeup signal of its own.

An example of the communication is shown in figure 5.8. First node A has data to send. It will immediately trigger a listen period for detecting any ongoing transmissions. Since none is detected, it will assume the role of interrogator and start sending a wakeup signal. Node C is out of its range and experiences only noise. Node B on the other hand is in the range and will detect the wakeup signal on its next listen period assuming the role of a tag. It receives the data at the end of the wakeup signal and replies at its chosen time slot. Node A has no more data and sends a *sleep* command to Node B to terminate the communication.

After the next listen period Node B in turn assumes the role of interrogator. Nodes A and C accidentaly choose the same reply slot and the replies collide. Node B reacts by resending the data after the reply time window has expired. Now nodes A and C choose different reply slots and node B receives their acknowledgements. Finally, node B issues a *sleep* command to the nodes separately. If the original packet was not addressed to node C, it will continue the cycle until the message reaches the destination.

Two optimizations were made to decrease the energy consumption of the nodes. First, if a tag receives a point-to-point message not addressed to it immediately after the wakeup signal, it can resume the sleep state until the beginning of its next wakeup period. Any following transmissions on will only take place between the interrogator and the addressed tag. Second, the number of failed collection rounds before the interrogator resumes the sleep state can be configured to other than three.

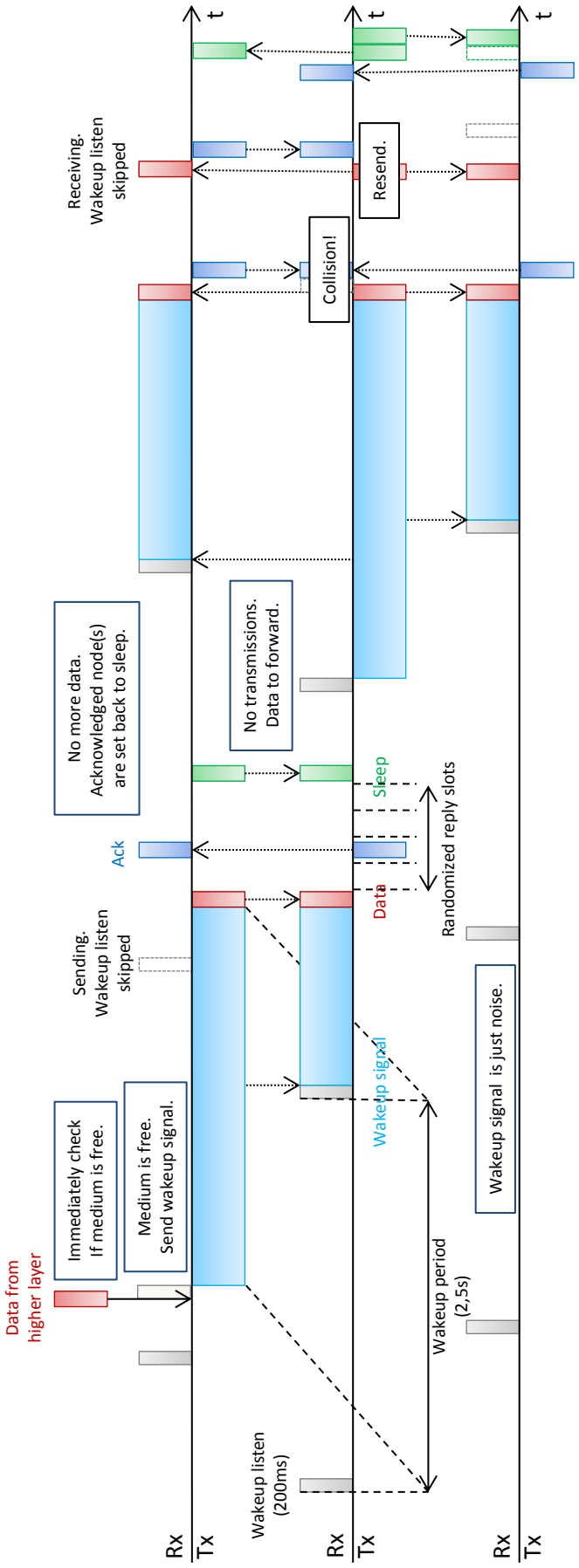


Figure 5.8: ISO-MAC communication example with three nodes.

The NED description of the ISO-MAC layer is given in Listing 5.2. The timing can be configured by altering the values of *wakupInterwal*, *wakupListenTime* and *wakeupSendTime*. The first listening period occurs *wakeupOffset* after the simulation starts. By using a randomized value it is very unlikely that two neighbouring nodes begin their listen period and thus their transmissions at the same point in time. As in the INET framework, the ISO-MAC gets the notification of changes in radio state through the notification board module. There must be exactly one notification board for each radio. In dual protocol stack implementation there are two notification boards and therefore ISO-MAC uses parameter *notificationBoardName* to identify which one of the modules to subscribe to.

```
simple Iso180007BasicMac like INodeLayer
{
    parameters:
        @class(Iso180007BasicMac);
        @display("i=block/layer;is=n");
        // Wake up interval, standard is 2.5-2.7s.
        double wakeupInterval @unit(s) = default(2.5s);
        // Listen time for wake up, standard is 200ms.
        double wakeupListenTime @unit(s) = default(200ms);
        double wakeupSendTime @unit(s) = default(this.wakeupInterval);
        double wakeupOffset @unit(s) =
                default(uniform(0s,this.wakeupInterval));
        // Initial reply time window, standard is 57.3ms.
        double windowSize @unit(s) = default(57.3ms);
        int nodeId = default(uniform(0,1024)); //Interpret as tagId
        int ownerId = default(0);
        int maxRetries = default(3); //Retries for unresponded messages
        int maxResends = default(4); //Retries for collided acks
        int repairThreshold = default(3); //Threshold for link repair
        string notificationBoardName = default("notificationBoard");

    gates:
        input upperIn @labels(RoutingMsg);
        output upperOut @labels(RoutingMsg);
        input lowerIn @labels(Iso180007Msg);
        output lowerOut @labels(Iso180007Msg);
        inout proc @loose;
}
```

Listing 5.2: ISO-MAC layer description in NED.

The Finite State Machine (FSM) implementing the ISO-MAC protocol is given in Figure 5.9. The functionality is logically divided into reception and transmission. In both cases the cycle begins in the *idle* state where the radio is turned off.

Periodically after *wakeupInterval* has passed a timer triggers a state change from

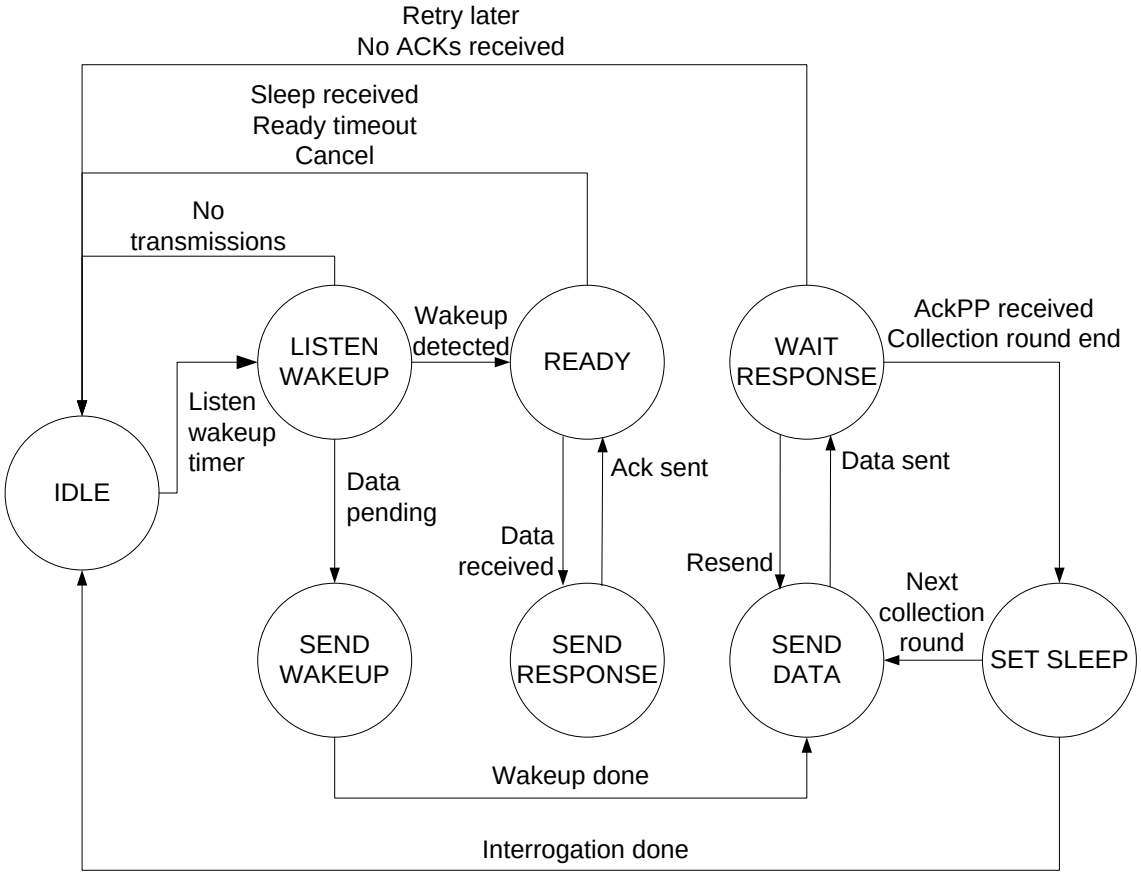


Figure 5.9: ISO-MAC operation as a finite state machine.

to *listen wakeup*. The radio is turned on to listen to the channel for *wakeupListenTime*. If no wakeup signal is detected, the ISO-MAC returns to *idle* and radio is turned back off. However, if an ongoing wakeup signal is detected, the ISO-MAC changes to *ready* state to wait for the command message. After receiving the command message ISO-MAC will wait in the *send response* state until the beginning of its designated reply slot. During the reply slot it sends the response message and returns to *ready* state for the next command. Receiving a *sleep* command activates the transition back to *idle* state. A timeout period was added to return the ISO-MAC to the *idle* state should the interrogator fail to send or the link break. If ISO-MAC does not receive a message during a timeout period it returns from *ready* state to *idle* state. The duration of the timeout period is equal to the *wakeupInterval* period.

If there are messages pending in the transmit buffer, the node may assume the role of an interrogator after the listen period in *listen wakeup* state. A wakeup signal is sent for *wakeupSendTime* in *send wakeup* after which the command is transmitted in the *send data* state. Next, the interrogator will wait in *wait response* state for the period of the reply window defined by *windowSize*. It stores the tag replies

in a response buffer and records possible collisions. If the data message was of a point-to-point type, there will only be a maximum of one response whose reception will trigger a transition to *set sleep* and intialize the sending of *sleep* command. If the message was a broadcast, the transition is triggered by a timer after the whole window has expired. All the tags whose reply was successfully received and stored is now set to sleep with a point-to-point *sleep* command. Any record of collisions results in an interrogation retry after transition to *send data* and the commmand is resend to any remaining tags. After all or no tags reply or the number of retries equals *maxRetries*, the interrogator will return to *idle* state.

5.3 Simulation

The simulation is set up with ten devices spread over a 100x40 m area. Five of the devices have only the low-power protocol stack and a motion sensor. Four devices have both the low-power and the WLAN protocol stacks. One is selected as a gateway device. Finally, one additional router without sensors is added to provide a route from the furthest surveillance device to the gateway. The setup is shown in Figure 5.10. The motion sensor ranges are marked with black circles around the sensing nodes. One motion stimulus is set to move horizontally back and forth to trigger the motion sensors repetively over the simulation.

The routes from the security nodes to the gateway are initially configured in the routing tables. SecurityNode 0 and SecurityNode 1 communicate directly with the Gateway and SecurityNode 2 through RouterNode 0 and SecurityNode 1. The low-power nodes do not initiate route discoveries since all alarm messages are sent as broadcast.

Both the protocol stacks use the 2.4GHz frequency channel, but on separate channel. The low-power radio module configuration is based on Texas Instruments

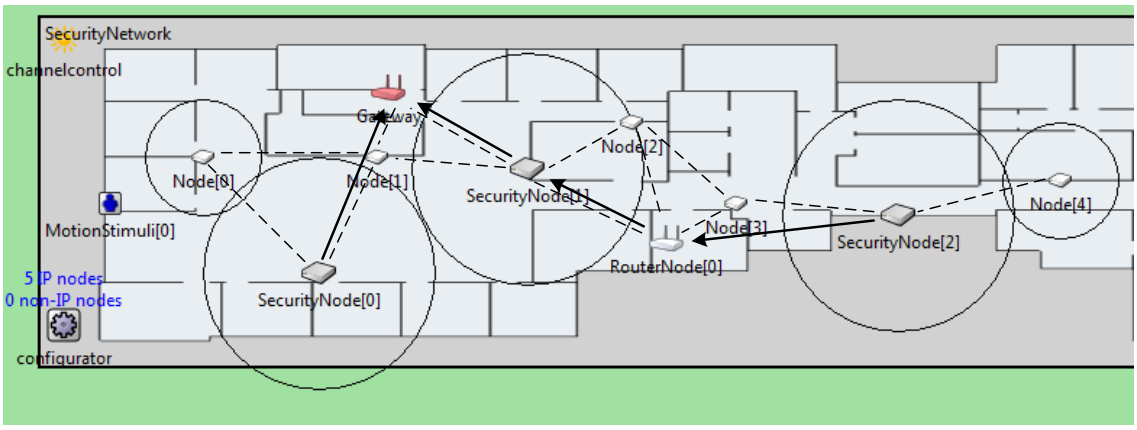


Figure 5.10: Simulation setup captured from OMNeT++ Tkenv. The possible links between the nodes are also marked.

Table 5.1: Energy consumption estimates of the devices.

Device	**WLAN average energy consumption [mW]**	**Low-power radio average energy consumption [mW]**	**Average communication energy consumption [mW]**
SecurityNode 0	283.9	21.6	305.5
RouterNode 0	297.6	21.8	319.4
Node 0	n/a	21.8	21.8

CC2530 [45] chip. It is designed for IEEE802.15.4 and ZigBee compliant communication with low energy consumption and provides a plausible reference for the model parameters. The nominal currents are 29.0 mA for transmission, 24.0 mA active reception and 0.4 μA for the most efficient power saving mode. Assuming the maximum operation voltage of 3.6 V, the power consumptions are 104.40 mW, 86.40 mW, and 2.88 μW respectively. The radio communication range is about 15 meters.

The WLAN radio configuration is based on Texas Instruments SimpleLink CC3000 [46] WLAN radio chip. The nominal currents are 190.0 mA for transmission, 92.0 mA for reception and 5.0 μA for idle state. The operation voltage is 3.6 V. The power consumptions are 684.0 mW, 331.2 mW, 18.0 μW respectively. The WLAN communication range is up to 49 meters.

The simulation was run for one hour of simulation time. The ISO-MAC access cycle was adjusted to 1.25 seconds, half of the minimum in the ISO-18000-7 standard, to lower the latency. The cost is an increase in energy consumption, since the nodes will have to wake up to listen to the channel more often. The simulated minimum latency for alarm messages is 1.46 seconds. On average the alarm latency is 17.00 seconds and the maximum measured latency is 40.67 seconds.

The average power consumption estimates for different devices are summarized in Table 5.1. WLAN communication consumes over 10x power compared to the low-power radio. Without the lock module, the difference would be even greater.

The evaluation of the results would benefit from comparison with a prototype or a mathematical model. While there is no documentation of the prototype energy consumption, the results can be compared with a TUTWSN node. Furthermore, additional simulations with varying number of devices, data rates and access cycle times would provide better coverage and lead to a refined model.

The energy consumption of a TUTWSN node is 60.17 mW for active reception, up to 42.17 mW for transmission and 37 μW for sleep mode [15]. The values have been measured with 3.0 V supply voltage and includes the power consumption of an active MCU. Compared to TUTWSN the simulated power consumptions are in the correct order of magnitude. However, since the simulated models omit the MCU,

Table 5.2: Total number of lines of C++ code to implement the model behaviour.

Module	**Lines of code**
Motion Sensor	473
Surveillance Application	601
AODV Routing	824
Routing Table for AODV	246
ISO-MAC	1729
WLAN Lock	158
Total	**4031**

the estimated energy consumptions are likely too optimistic. Even with average energy consumption of 21.8 μW, battery-operated node devices are not applicable in WSNs. However, the difference of over 10x compared to WLAN suggests a significant improvement in energy consumption. The means to simulate multiple radios in a single device is considered more important result in this work than the absolute energy consumption values.

In TUTWSN the access cycle is 2 seconds. On average each hop has 1 s latency. The simulated minimum latency of 1.46 s is comparable to a single hop latency. Since the interrogator uses 0.2 s for channel listening and 1.25 s for sending a wakeup signal, the latency can not be further decreased without lowering the access cycle time. The simulated network is very small in the scale of WSNs and based on the average latency the MAC and routing protocols may not be applicable in larger networks.

The modeling effort can be estimated by the amount of source code lines since no estimates of the time usage are available. Table 5.2 shows the number of lines of C++ code for each module implementation. The amount includes all the inherited classes and header files. As seen in Listings 5.1 and 5.2 the NED files contain only few lines and the total amount is estimated to about 200 lines.

6. MOBILE POSITIONING NETWORK

The objective of the second simulation case was to evaluate the accuracy of a positioning algorithm in WSNs. Traditionally global positioning system (GPS) has been used for device positioning, but in energy-constrained WSNs, it is not viable energy- or cost-wise for each node to have a GPS receiver.

This chapter shows a simulation for evaluating one possible locationing algorithm in WSNs. The algorithm was initially evaluated using a Python script for simulating the node mobility and communication. Importing the simulation to OMNeT++ provides more options for configuring the scenario. It enables more diverse mobility models to be used and improved simulation of the physical environment.

6.1 Positioning Algorithm

In [47] Shang et. al propose a positioning algorithm utilizing connectivity information already available in most WSNs. In a fully connected network the relative positions of the nodes can be estimated from the distances between the nodes. The distances can be mathematically estimated using the Received Signal Strength Indication (RSSI) values.

The estimate results in a map which is arbitarily rotated and flipped as the node locations are relative to each other and not absolute. However, with only a few anchor nodes equipped with a GPS the map can be transformed to the absolute node positions. The anchor nodes can be equipped with an external power source to provide the node with sufficient energy. The used mathematical method, Multi-Dimensional Scaling (MDS), is too computationally intensive to be executed in the WSN network, but can be performed in an external server. Consequently, the nodes will have to send the collected statistics to the server gateway for analysis.

6.2 Implementation

OMNeT++ was used to simulate the node movement and communication. The nodes record the RSSI value for each received messages. The absolute position is recorded for reference and used to calculate the error of the position estimates. The MDS positioning algorithm is run in Matlab using the record files as input. A Python script was used to produce other set of input records for comparison. The simulation flow is shown in Figure 6.1.

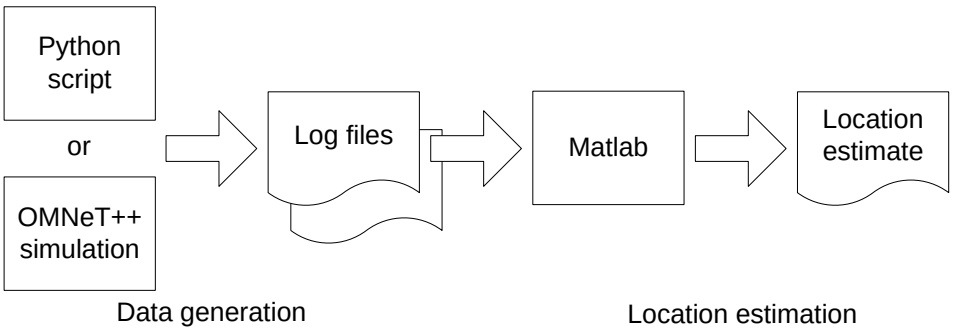


Figure 6.1: Positioning simulation flow.

Compared to the Python script, OMNeT++ simulation targets to provided more realistic model by introducing new error sources accounted in real WSN environments. The messages may be lost or collide in the wireless medium and all the transmissions have a non-constant latency which was not accounted for in the Python implementation. Modeling these physical phenomena in Matlab would be cumbersome. Conversely, running a complex computation algorithm in Matlab is more convenient than modeling it in OMNeT++.

The simulation requires mobile nodes with position and RSSI logging. The mobility is enabled by using the Mobility module provided by INET framework. The communication is handled by the MAC and routing protocols described in the previous chapter. The INET radio model used as a base for the low-power protocol stack radio already calculates the RSSI values for the received packets. However, the other layers have no access to it and therefore the radio model has to be modified.

In a real implementation, the radio would store the RSSI values into memory and the application would later read, process and send it to the gateway. In the simplified simulation model, the radio was modified to write the RSSI value directly into the application payload data. The application layer records the RSSI values of the received messages into a file when the payload data is delivered to it through the protocol stack.

The distance between two nodes is estimated using three methods. The methods are visualized in Figure 6.2. The first method, *real distance*, calculates the absolute distances from the recorded node positions. The second method, *sum distance*, estimates the distance between any two nodes as the sum of the distances of the intermediating nodes. The distance between two neighbouring nodes are estimated from the RSSI value. This method is simple, but generally tends to yield longer distances than the real distance. The final method, *hop distance*, uses the number of hops between the nodes as the distance.

The simulation does not take into account the delay from node to gateway or that some messages that are likely to drop before reaching the gateway. The application

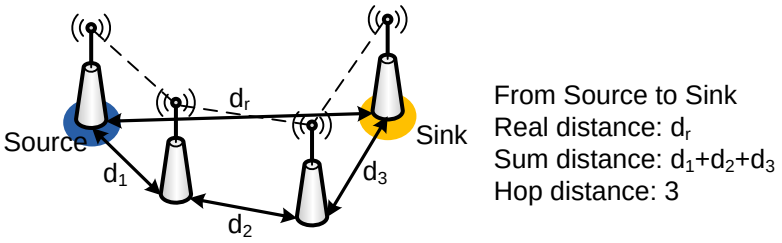


Figure 6.2: Three estimation methods for node distances in multihop.

layer needs to broadcast messages periodically to generate traffic in the network. Broadcasting eliminates the cost for route building and maintenance.

6.3 Simulation

The network has 20 nodes including 4 anchor nodes. The nodes move with a constant speed and change their direction every 5 seconds. The radios operate on 433 MHz frequency band and the communication range is set to maximum of 105 meters. Each node broadcasts randomly between 6 to 8 seconds. The following sections describe three simulation scenarios. The simulation time in all scenarios is 10 minutes.

6.3.1 Group Walk

In the first scenario all the nodes move in a 10x10 km area as a loose group. Initially the nodes move around the area randomly. The node with the lowest id acts as a group leader. If any other node detects that no neighbours are within its communication range, it will immediately turn towards the group leader to rejoin the group. Figure 6.3 shows an example of the node behaviour. The node movement in this case is mimics the Python implementation.

The measured average location errors for different estimates are shown in Table 6.1 and visualized in Figure 6.4. Using the absolute distances the average location error is measured in centimeters. At worst it is less than 0.2% of the communication range so the error introduced by the MDS algorithm is very small.

Using the other two methods the error increases almost linearly with the node speed. The location estimate seems to be most accurate when the nodes move at a very slow speed. Only after the node speed exceeds 0.50 m/s the error is greater than with stationary nodes. It is not surprising that using only the number of hops as the distance, the average location error is always higher than with the sum estimate.

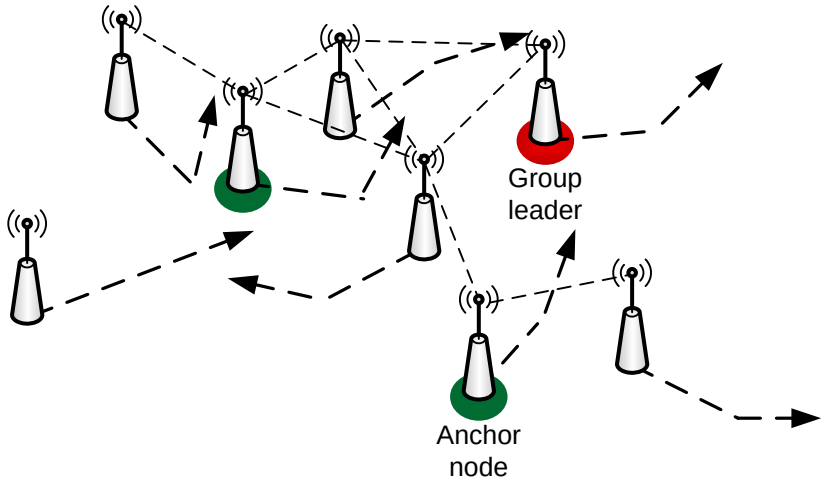


Figure 6.3: In the group walk scenario the nodes follow the leader node.

Table 6.1: Average location error in group walk simulation.

Node Speed [m/s]	**Average Error of Real Distance [m]**	**Average Error of Sum Distance [m]**	**Average Error of Hop Distance [m]**
0.00	0.02	59.41	74.34
0.10	0.02	26.79	48.00
0.20	0.01	39.62	62.61
0.30	0.02	61.13	84.16
0.40	0.01	39.09	61.13
0.50	0.01	54.71	75.64
0.60	0.02	60.28	81.18
0.70	0.01	81.23	101.04
0 80	0.01	48.75	63.50
0.90	0.01	66.15	82.39
1.00	0.01	58.25	75.27
1.10	0.01	67.74	88.51
1.20	0.01	74.58	78.46
1.30	0.01	108.08	108.31
1.40	0.01	68.49	88.05
1.50	0.01	86.81	98.94
1.60	0.01	75.51	86.03
1.70	0.01	79.78	90.44
1.80	0.01	85.96	98.16
1.90	0.01	85.45	103.78
2.00	0.02	86.79	91.12

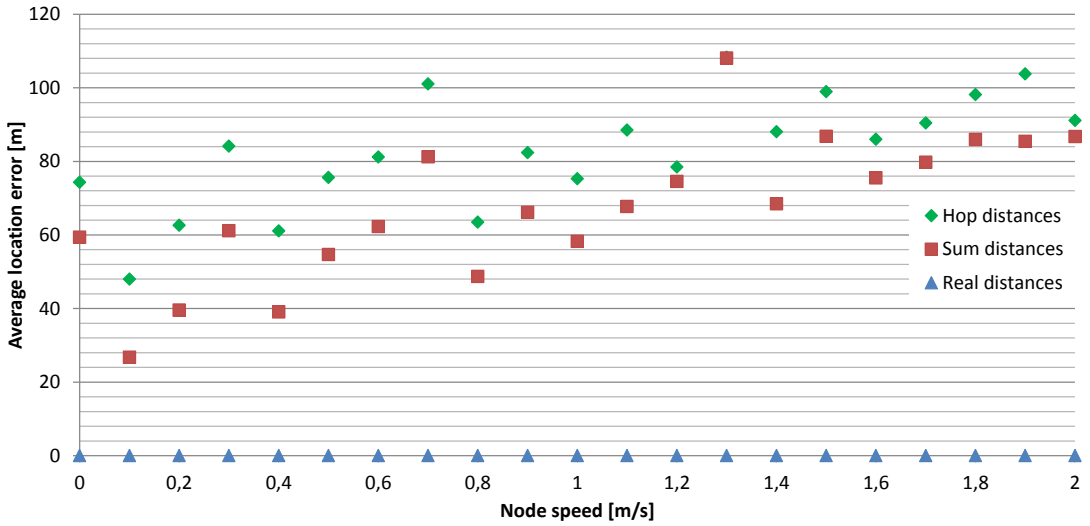


Figure 6.4: Node speed vs. average location error in group walk.

6.3.2 Stationary Anchor Nodes

The second scenario corresponds to indoor positioning. The nodes are limited to 300x300 m area and the anchor nodes remain stationary. All other nodes move randomly in the area. The anchor nodes have been located to cover the whole area so that the mobile nodes have always a link to at least one of them. The scenario is shown in Figure 6.5.

The average location errors for different estimate methods are given in Table 6.2 and visualized in Figure 6.6. Compared to the group walk the average location error is smaller in most cases especially if all the nodes are stationary.

Although the location error is smaller than in the group walk scenario, if the node

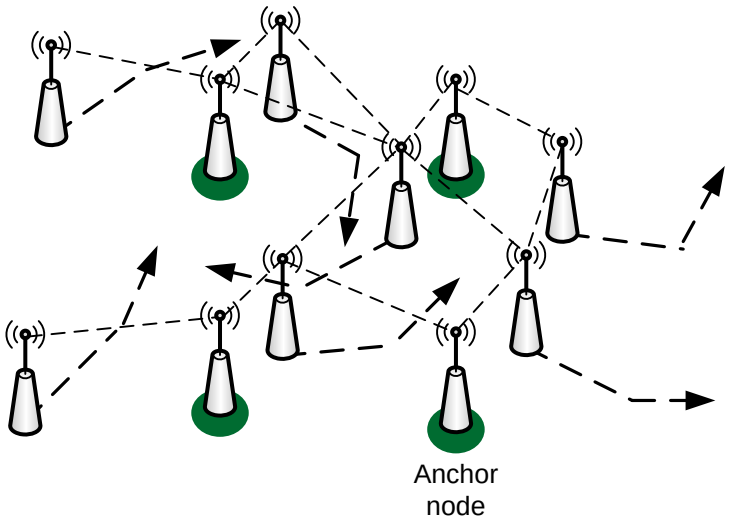


Figure 6.5: The anchor nodes remain stationary while other nodes move freely in the area.

Table 6.2: Average location error in stationary anchor nodes simulation.

Node Speed [m/s]	Average Error of Real Distance [m]	Average Error of Sum Distance [m]	Average Error of Hop Distance [m]
0.00	0.56	30.39	57.05
0.10	1.04	35.40	63.21
0.20	0.93	28.69	53.92
0.30	0.43	33.64	53.95
0.40	0.64	39.18	69.91
0.50	0.58	38.97	59.34
0.60	1.54	38.66	60.85
0.70	0.79	48.90	94.29
0 80	0.95	50.15	73.32
0.90	2.24	48.64	76.12
1.00	1.48	50.90	73.14
1.10	0.63	53.08	70.95
1.20	1.29	60.78	79.19
1.30	1.68	60.58	77.58
1.40	1.83	60.33	81.15
1.50	0.87	65.71	85.67
1.60	1.11	62.35	84.97
1.70	1.74	63.76	81.74
1.80	1.66	62.68	86.67
1.90	2.08	71.59	90.63
2.00	1.32	62.66	86.79

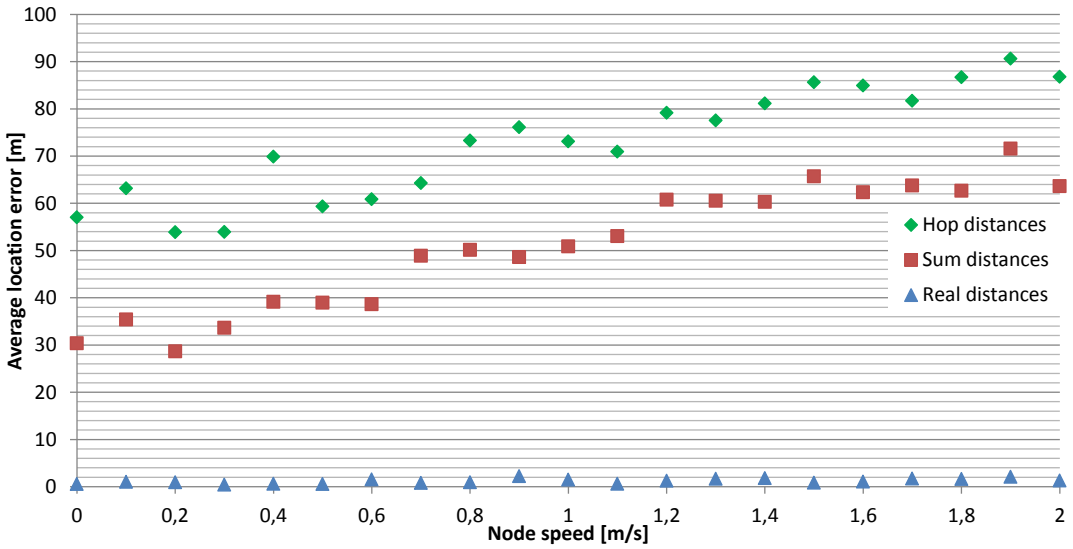


Figure 6.6: Node speed vs. average location error with four stationary anchor nodes.

speed is over 1 m/s, the average error is still over 50% of the communication range. This may be acceptable for some applications for example in indoors positioning where the required accuracy is at room level.

6.3.3 Constant Speed Walk

In the final scenario, all the nodes move in the same direction with the same constant speed of 2 m/s, but the maximum number of hops in the network is altered. The initial positioning of the nodes designate the maximum number of hops. First, all nodes are grouped tightly and every node is within the communication range of the others. Then the nodes are spread out until finally the nodes form a single line and each node can only communicate with the previous and the next node. This is illustrated in Figure 6.7. In the final stage, the maximum number of hops is 19.

The simulation results for are shown in Table 6.3 and Figure 6.8. Location error increases almost linearly with the hop count. The sole exception is at the maximum number of hops is four. This is probably caused by positioning all the anchor nodes on the same line which makes the estimation by the MDS algorithm difficult. For over 8 or more hops the hop distance is no longer applicable as the error exceeds 100% of the communication range.

Three simulation scenarios were used to evaluate the MDS algorithm for WSN positioning. Based on the simulation results, the MDS algorithm would be applicable to location-based applications with loose accuracy requirements such as indoor

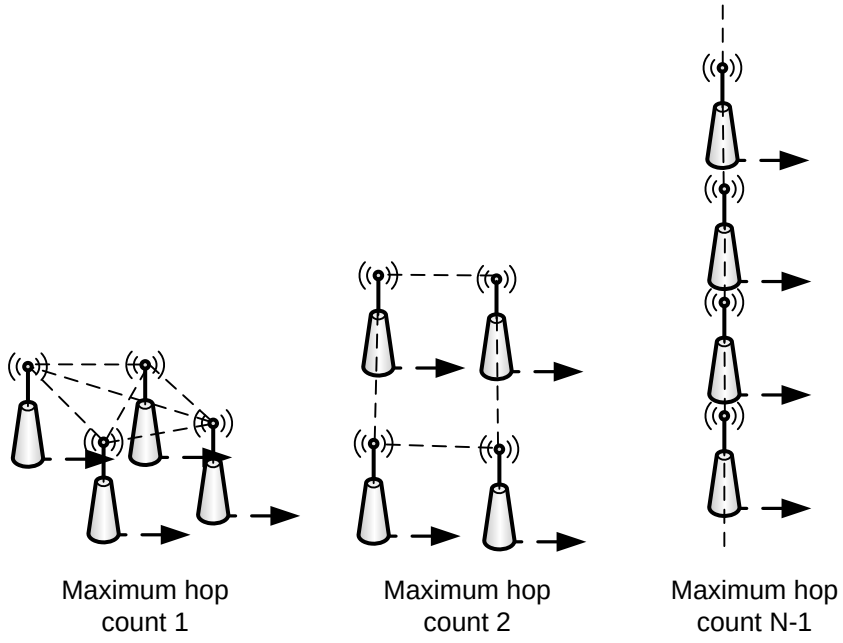
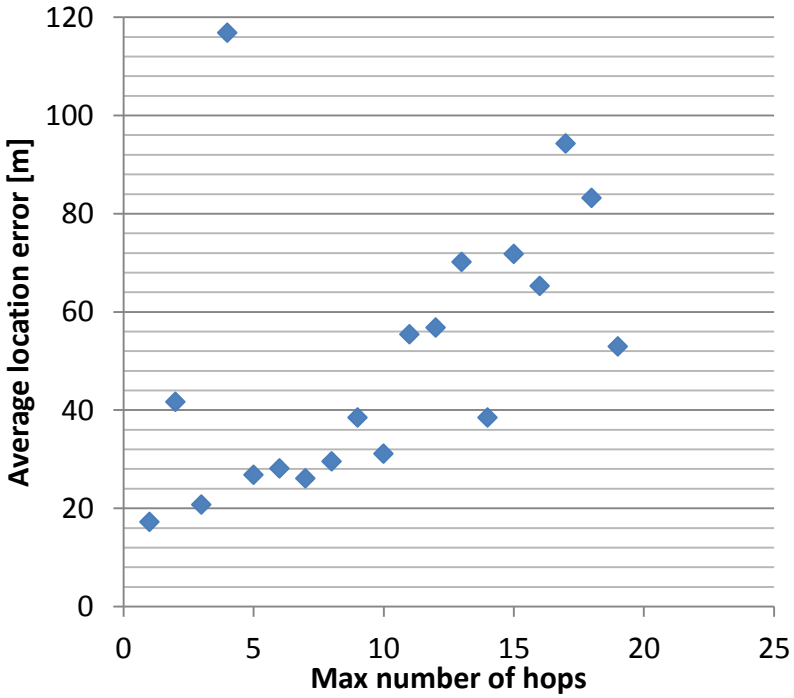


Figure 6.7: Maximum number of hops in the network was altered by spreading out the nodes.

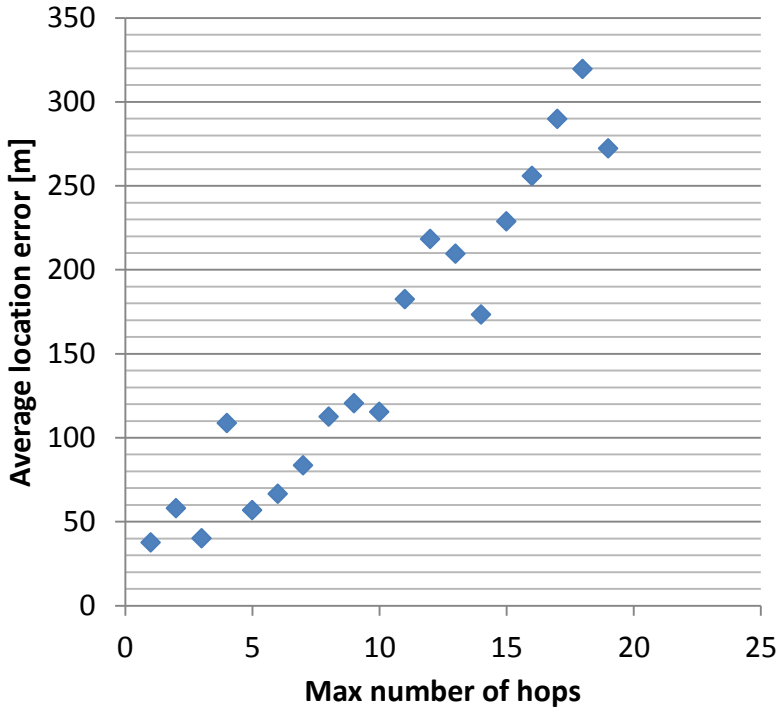
Table 6.3: Average location error for maximum number of hops in the network.

Maximum Number of Hops	**Average Error of Sum Distance [m]**	**Average Error of Hop Distance [m]**
1	17.24	37.60
2	41.71	57.90
3	20.73	40.01
4	116.84	108.70
5	26.84	56.80
6	28.10	66.50
7	26.10	83.60
8	29.57	112.50
9	38.52	120.50
10	31.14	115.30
11	55.42	182.50
12	56.82	218.30
13	70.16	209.60
14	39.51	173.30
15	71.82	228.80
16	65.32	255.90
17	94.31	289.80
18	83.23	319.60
19	52.95	272.30

positioning. The location error is less than 50% of the communication range for devices moving slowly. For low-power devices with small communication range this is often sufficient. Even simple estimates of the node distances were found to yield decent estimates.



(a) Number of hops vs. location error estimated by sum distances.



(b) Number of hops vs. location error estimated by hop counts.

Figure 6.8: The location error increases almost linearly with the maximum number of hops in the network.

7. CONCLUSIONS

The objective of this Thesis was to deploy OMNeT++ simulation environment in WSN research. Two simulation cases were implemented to evaluate the simulator and modeling effort. These cases required new simulation models to be designed since no existing models fullfilled the case requirements.

The simulation environment was deployed and the simulations were run successfully. The simulations showed that hierarchical operation and alarming can reduce the overall energy consumption in the network. However, the alarm latency will probably not scale well with the networks size, if the modeled MAC and routing protocols are used.

The challenge of a layered protocol stack design is the unavoidable dependencies between the layers and cross-layer communication. The implementation accepts this coupling of layers and does not try to provide a general layer interface that can be connected to any other layer. The inevitable consequence is that the models are incompatible with any other existing models. However, it simplifies the design considerably.

OMNeT++ has proven to be a stable, easily extensible, and scalable simulator. It can be considered in WSN research. Althought fewer models are available compared to many other simulators, many projects are working to extend the available model libraries. Current simulation models often show too optimistical results. As the libraries evolve, new reference models become available and the existing models will become more accurate. Also the modeling effort reduces as the nodes can be assembled from interchangable layers according to the application requirements instead of modeling all the required layers.

Future work of the models begins with the decoupling of the protocol layers. The functionality can be further optimized and improved in many aspects. For example, the surveillance application would benefit from clustering the network into surveilance areas that can be individually alerted. The neighbouring areas are alerted only if the event is likely to move to that area. WSN research would benefit from more accurate energy and resource consumption estimation. As the work on network simulators proceeds, new and more detailed models will hopefully emerge.

REFERENCES

[1] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002.

[3] J. Sokolowski and C. Banks, *Principles of Modeling and Simulation: A Multidisciplinary Approach*. Wiley, 2011.

[4] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Wiley Publishing, 2010.

[5] C. Singh, O. Vyas, and M. Tiwari, "A survey of simulation in sensor networks," in *Computational Intelligence for Modelling Control Automation, 2008 International Conference on*, pp. 867–872, Dec 2008.

[6] "Omnet++ network simulation framework." [WWW]. Accessed 31.3.2014. `http://www.omnetpp.org/`.

[7] B. Stroustrup, *The C++ Programming Language, 4th Edition*. Addison-Wesley Professional, 4 ed., May 2013.

[8] E. Callaway, *Wireless Sensor Networks: Architectures and Protocols*. Internet and Communications, Taylor & Francis, 2004.

[9] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks." RFC 4944 (Proposed Standard), Sept. 2007.

[10] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification." RFC 2460 (Draft Standard), Dec. 1998.

[11] "IEEE standard for Local and metropolitan area networks–part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pp. 1–314, Sept 2011.

[12] M. J. S. Smith, *Application-Specific Integrated Circuits*. Addison-Wesley Professional, 1st ed., 2008.

[13] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA, USA: California Technical Publishing, 1997.

[14] Xilinx: What is an FPGA? Field Programmable Gate Array (FPGA) [WWW]. [Accessed 31.3.2014]. http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm.

[15] M. Kuorilehto, M. Kohvakka, J. Suhonen, P. Hamalainen, M. Hannikainen, and T. D. Hamalainen, *Ultra-Low Energy Wireless Sensor Networks in Practice: Theory, Realization and Deployment*. Wiley Publishing, 2008.

[16] IEEE Std 802.11-2007, "IEEE standard for information technology — telecommunications and information exchange between systems — local and metropolitan area networks — specific requirements — part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," June 2007.

[17] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-demand Distance Vector (AODV) routing," 2003.

[18] ISO/IEC, "Information technology - Radio frequency identification (RFID) for item management - part 7: Parameters for active air interface communications at 433Mhz," ISO 18000-7, International Organization for Standardization, Geneva, Switzerland, 2004.

[19] ZigBee Alliance, "Zigbee specifications," 2005.

[20] M. Weyn, G. Ergeerts, L. Wante, C. Vercauteren, and P. Hellinckx, "Survey of the DASH7 alliance protocol for 433 Mhz wireless sensor communication," *International Journal of Distributed Sensor Networks*, vol. 2013.

[21] L. G. Roberts, "Aloha packet system with and without slots and capture," *SIGCOMM Comput. Commun. Rev.*, vol. 5, pp. 28–42, Apr. 1975.

[22] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, (New York, NY, USA), pp. 56–67, ACM, 2000.

[23] A. Varga, "OMNeT++ user manual." [WWW]. Accessed 12.2.2013. http://www.omnetpp.org/doc/omnetpp/Manual.pdf.

[24] A. Sobeih, W. peng Chen, J. C. Hou, L. chuan Kung, N. Li, H. Lim, H. ying Tyan, and H. Zhang, "J-sim: A simulation and emulation environment for wireless sensor networks," *IEEE Wireless Communications magazine*, vol. 13, p. 2006, 2005.

[25] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks," in *Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on*, pp. 154–161, May 1998.

[26] "NS-2 network simulator." [WWW]. Accessed 13.10.2013. `http://www.isi.edu/nsnam/ns/`.

[27] "NS-3 network simulator." [WWW]. Accessed 13.10.2013. `http://www.nsnam.org/`.

[28] S. Sundresh, W. Kim, and G. Agha, "Sens: A sensor, environment and network simulator," in *Proceedings of the 37th Annual Symposium on Simulation*, ANSS '04, (Washington, DC, USA), pp. 221–, IEEE Computer Society, 2004.

[29] S. Park, A. Savvides, and M. B. Srivastava, "Sensorsim: A simulation framework for sensor networks," in *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '00, (New York, NY, USA), pp. 104–111, ACM, 2000.

[30] Network Planning and Configuration | Network Simulation (OPNET Modeler Suite) | Riverbed [WWW]. Accessed 23.3.2014. `http://www.riverbed.com/products-solutions/products/network-performance-management/network-planning-simulation/Network-Simulation.html`.

[31] Simulink - Simulation and Model-Based Design [WWW]. Accessed 23.3.2014. `http://www.mathworks.se/products/simulink/`.

[32] NetSim Cisco Network Simulator & Router Simulator [WWW]. Accessed 23.3.2014. `http://www.boson.com/netsim-cisco-network-simulator`.

[33] J. Gosling, B. Joy, G. Steele, and G. Bracha, *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.

[34] The Perl Programming Language [WWW]. Accessed 23.3.2014. `http://www.perl.org/`.

[35] J. K. Ousterhout, "Tcl and the tk toolkit," 1994.

[36] G. Rossum, "Python reference manual," tech. rep., Amsterdam, The Netherlands, The Netherlands, 1995.

[37] R. Bagrodia, M. Takai, Y. an Chen, X. Zeng, and J. Martin, "Parsec: A parallel simulation environment for complex systems," *IEEE Computer*, vol. 31, pp. 77–85, 1998.

[38] The Eclipse Foundation. Eclipse Integrated Design Environment [WWW]. Accessed 23.3.2014. https://www.eclipse.org/.

[39] "Inet framework." [WWW]. Accessed 31.3.2014. http://inet.omnetpp.org/.

[40] "Mixim." [WWW]. Accessed 31.3.2014. http://mixim.sourceforge.net/.

[41] "Inetmanet." [WWW]. Accessed 31.3.2014. https://github.com/aarizaq/inetmanet-2.0.

[42] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, pp. 79–84, May 2007.

[43] A. Boulis, "Castalia - A simulator for Wireless Sensor Networks and Body Area Networks. User's Manual," Mar. 2011.

[44] T. van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, (New York, NY, USA), pp. 171–180, ACM, 2003.

[45] Texas Instruments. A True System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee Applications. TI CC2530 Datasheet [WWW]. Accessed 3.4.2014. http://www.ti.com/lit/ds/symlink/cc2530.pdf.

[46] Texas Instruments. TI SimpleLink CC3000 Module Datasheet [WWW]. Accessed 3.4.2014. http://www.ti.com/lit/ds/symlink/cc3000.pdf.

[47] Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz, "Localization from mere connectivity," in *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc '03, (New York, NY, USA), pp. 201–212, ACM, 2003.